Mission Research Corporation

MRC/WDC-R-445

MAGIC USER'S MANUAL

Larry Ludeking
David Smithe
Mike Bettenhausen
Scott Hayes

March 1999

Technical Report

Prepared for: Air Force Office of Scientific Research
Bolling Air Force Base
Washington, D.C. 20332-6448

Contract No.: F49620-96-C-0032

Prepared by: MISSION RESEARCH CORPORATION
8560 Cinderbed Road
Suite 700
Newington, Virginia 22122

Research sponsored by the Air Force Office of Scientific Research under Contract F49620-96-C-0032. The United States Government is authorized to reproduce and distribute reprints for Governmental purposes nothwithstanding any copyright notation hereon.

AFRL-SR-BL-TR-99-

0243

REPORT DOCUMENTATION PAGE

1. AGENCY USE ONLY (leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1999	Final 1 July 1996 through 31 October 1999

4. TITLE AND SUBTITLE
MAGIC USER'S MANUAL

5. FUNDING NUMBERS

6. AUTHOR(S)
Larry Ludeking, David Smithe, Mike Bettenhausen, and Scott Hayes

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Mission Research Corporation
8560 Cinderbed Road, Suite 700
Newington, VA 22122-8560

8. PERFORMING ORGANIZATION REPORT NUMBER
MRC/WDC-R-445

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Office of Scientific Research
801 N. Randolph Street, Room 732
Arlington, VA 22203-1977

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12b. DISTRIBUTION/AVAILABILITY STATEMENT
Distribution Statement A. Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT
Current research efforts in the area of high-power microwave and plasma physics make extensive use of computer simulation to reduce design time, investigate new design concepts, and examine the performance envelope of a wide variety of devices. The research includes, but is not limited to, such diverse items as SWO's, TWT's, klystrons, gyrotrons, the helicon, CFA's, and micro-hollow cathodes. The instrument developed for much of this research is the MAGIC Tool Suite. MAGIC is a general purpose electromagnetic particle-in-cell code, i.e., a finite-difference, time-domain code for simulating plasma physics processes, i.e., those processes that involve interactions between space charge and electromagnetic fields. As a result, the application suite is applicable to broad classes of plasma physics problems. The software suite includes powerful algorithms to represent structural geometries, material properties, incoming and outgoing waves, particle emission processes, and so forth. It has a mature and robust package of diagnostic features that have made it the tool of choice for many universities and laboratories.

14. SUBJECT TERMS
Microwave, simulation, cross-field amplifier, TWT, klystron, BWO, helicon, gyrotron

15. NUMBER OF PAGES
349

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18
298-102

MRC/WDC-R-445
Copy 49

MAGIC USER'S MANUAL

Larry Ludeking
David Smithe
Mike Bettenhausen
Scott Hayes

March 1999

Technical Report

Prepared for: Air Force Office of Scientific Research
Bolling Air Force Base
Washington, D.C. 20332-6448

Contract No.: F49620-96-C-0032

Prepared by: MISSION RESEARCH CORPORATION
8560 Cinderbed Road
Suite 700
Newington, Virginia 22122

Research sponsored by the Air Force Office of Scientific Research under Contract F49620-96-C-0032. The United States Government is authorized to reproduce and distribute reprints for Governmental purposes nothwithstanding any copyright notation hereon.

TABLE OF CONTENTS

PART 1. USING MAGIC

PART 2. MCL COMMANDS

PART 3. TIME AND SPACE

PART 4. SPATIAL EXTENSIONS

PART 5. PROPERTIES

PART 6. ALGORITHMS

PART 7. OUTPUT

COMMANDS INDEX

Preface

This User's Manual documents the March 1999 versions of MAGIC2D and MAGIC3D. MAGIC2D is a two-dimensional code and MAGIC3D is its three-dimensional counterpart. The same Manual applies to both. You will encounter references to "2D simulations," which currently can be performed only with MAGIC2D, and to "3D simulations," which currently can be performed only with MAGIC3D. With this manual, we continue toward the goal of unification of the two codes. Virtually all the algorithms and material models are available in both MAGIC2D and MAGIC3D. While some features are currently available in only one of the codes, our goal is to make all features universal.

With this release of MAGIC, we have converted to a Microsoft Windows-based version of the software and are including an electronic version of the *MAGIC User's Manual* with our distribution. (Note that the electronic copy of the manual is only available for PC's using one of the MS Windows operating systems.) MAGIC on workstations no longer links to NCAR or DISSPLA. Instead, MRC supplies a public domain, mbplot, which was developed by Mike Bettenhausen. Our User's Manual has been rewritten to reflect various command changes, new models and algorithms, and the altered operating environment.

New Models and Algorithms

Several new features have been added to MAGIC since the previous release. These include:

- An IONIZATION model was added to MAGIC2D. – There are two ionization models. The first allows the user to define an arbitrary source term and the seconds allows for impact ionization, where ion production is based on the cross section of the neutral target. In addition, this model includes collisional damping of electron kinematics.
- A revised FOIL transport model was added to MAGIC2D – The new FOIL scattering dynamics are courtesy of the Radiation Shielding Information Center (RSIC) at ORNL, and the Integrated Tiger Series (ITS) team. Interface routines were added to MAGIC to allow direct access to the source codes of XGEN and CYLTRAN. Only the electron transport portion of the CYLTRAN code was enabled. Work is in progress to add this to MAGIC3D.
- A SURFACE_LOSS model was added to MAGIC3D. This model computes losses due to surface currents on conducting surfaces.
- A RESONANT_PORT model was added to MAGIC2D. It applies an outer boundary that models the effect of a resonant cavity or lumped circuit. Work is in progress to add this to MAGIC3D.
- A MAXWELL QUASI_NEUTRAL algorithm was added. The quasi-neutral algorithm is expected to be most useful in a restricted class of problems in which magneto-hydro-dynamic (MHD) behavior is being investigated.
- The EMISSION EXPLOSIVE algorithm has been revised. The new model provides much better agreement with Child-Langmuir current over a broader voltage range. Note that for those who wish to use the previous model, it is still available by using EMISSION OLD_EXPLOSIVE.
- The FUNCTION command has had several new, predefined functions added. These include both complete and incomplete elliptic functions as well as two ramp functions for smooth initialization of signals. (These are particularly useful in the PORT and DRIVER commands.)
- A feedback circuit option was added to the PORT command. The PORT ... CIRCUIT option allows the user to specify the time-averaged current, power, or voltage that is to be supplied to a simulation through a simulation boundary.
- A feedback circuit option was added to the DRIVER command. The DRIVER ... CIRCUIT option allows you to specify the time-averaged current, power, or voltage that is to be supplied to a simulation from a specified current source.

- The PRESET command has been enhanced by the addition of a PANDIRA option, which allows the user to import magnetic fields from the POISSON/SUPERFISH codes into MAGIC.
- The IMPORT command has a new feature that permits the creation and import of a laminar beam assuming a confining Brillouin magnetic field. This would typically be used with the PRESET PANDIRA fields to introduce PPM fields.
- The MATERIAL command table has been extended to accommodate more material properties.
- The AREA command has several new shape options. These are FILLET, QUARTERROUND, and SINUSOID.

New Diagnostic Features

Several new diagnostic features have been added to MAGIC since the previous release. These include:

- On a PC, you can make movies by adding the MOVIE keyword to output diagnostics such as CONTOUR, PHASESPACE, RANGE, VECTOR and VIEW_3D.
- On a PC, you can make a BITMAP of a screen graphics display by pressing the F7 key.
- The OBSERVE command has been extensively upgraded to provide sensible measurement options for engineering and design. It includes the following types of measurement choices:

    - The OBSERVE FIELD_POWER command
        - The S.DA option gives the power for Poynting flow.
        - The E.J_DRIVER.DV option gives power converted from the prescribed driver current source.
        - The E.J_PARTICLE.DV option gives power converted from the interaction of the EM fields and the particle currents.
        - The E.J_OHMIC.DV option gives power absorbed in a conductance region.
        - The E.J_FREESPACE.DV option gives power absorbed in a free space region.
        - The SURFACE_LOSS option gives power absorbed on the surface of a conducting region.

    - The OBSERVE FIELD_INTEGRAL command
        - The E.DL option measures the voltage along a line integral.
        - The H.DL option measures the current around a line integral.
        - The J.DA option measures the net particle current through an area.

    - The OBSERVE PARTICLE_STATISTICS command provides a method of measuring a variety of particle statistical quantities, such as emittance.
    - The OBSERVE RESONANT_PORT command allows the user to measure the quantities associated with the RESONANT_PORT model.

- The RANGE command permits the measurement of quantities along one of the non-temporal axes in the simulation. New options for this command include.
    - The RANGE HISTOGRAM option allows you to make a histogram of particle distributions.
    - The RANGE FIELD_POWER option permits the measurement of power flow along a spatial axis.
    - The RANGE FIELD_INTEGRAL option allows integration of E.DL along one axis and permits the measurement profile to be displayed along another axis.

- The TABLE PARTICLES Command produces a data file that contains position, charge, and momentum for the particles in a specified area or volume.

MAGIC Tool Suite Manager

The MAGIC Tool Suite Manager is the browser for viewing the electronic MAGIC User's Manual. It allows the user to examine the manual by part or by individual command. In addition, it includes a set of example files that can be exercised directly from the Tool Suite Manager. The Tool Suite Manager will start MAGIC2D or MAGIC3D for the particular example selected. In addition, it allows the user to browse through the MAGIC tutorials that are also supplied in electronic form.

WEB Site

The MAGIC web site is located at "http://www.mrcwdc.com/Magic/Homepage.htm". At the web site a user can download the latest published updates to the manual as well as working versions of bug fixes. In addition, we publish announcements of interest to our users, including the time/date/place of the MAGIC User's Group meeting. Finally, we include links to our user sites. Please send an email if you wish to be added/removed from our site link list.

Acknowledgements

The authors would like to thank several people for their gracious help in preparing this document. In particular, Darlene Morrow and Nancy Parrish have provided immeasurable support in the editing and preparation of this material. In addition, we would like to thank Dr Bob Gray for his efforts in the development of the MAGIC Tool Suite Manager, the electronic manual browser. Finally, we would like to acknowledge our users, from whom we receive many requests and ultimately for whom we do this work, both for their support in beta testing and for their patience in giving us time to fix problems when they occur. Despite the invaluable assistance of all these persons and many others, we the authors are responsible for any omissions, errors, and inaccuracies that still remain. Continuing feedback from our users is an essential and invaluable resource and is greatly appreciated, both for the challenges they present and for the opportunities they provide to improve upon what we have already achieved. Thanks again to you all!

May 1999 *Lars Ludeking*

This page is intentionally left blank.

1. INTRODUCTION

1.1 MANUAL OBJECTIVE

This Manual documents the most recent release of the MAGIC2D and MAGIC3D codes. The codes and Manual are usually released annually. The version (month and year of release) is imprinted on the cover of this document and on virtually all output from the code. Interim versions of the code may be released without documentation to correct errors, to permit use of new algorithms, and to allow beta testing. Users who encounter errors in the code or Manual are encouraged to communicate these to Mission Research Corporation by fax at (703) 339-6953 or e-mail at magic@mrcwdc.com.

Insofar as it is possible, the code is designed to be backwards compatible with the Manual. This means that features in previous Manuals will probably work in the latest version of the code. Backwards compatibility is the reason that old input files can continue to run, even though many of the commands are technically obsolete and have been dropped from the Manual. The Manual reflects the future, while the code is a mixture of old and new features.

The Manual assumes that the reader may have no prior experience with the code, but that he is acquainted with computational physics methods in general and electromagnetic, particle-in-cell methods in particular. There is little attempt to motivate or even to justify the conventions and algorithms offered. Instead, the focus is upon how to use the code, and the user having previous experience with the code may wish to proceed to Part 2.

1.2 MANUAL ORGANIZATION

This Manual consists of seven Parts, and it is further subdivided into 24 Chapters. The names of the Part and Chapter are listed at the top of every page in the Manual.

Part 1 (Using the Code) is intended primarily for new users. It focuses on "how to" and is relatively devoid of theory and derivations. You are reading Chapter 1, which is an introduction. Chapter 2 describes how to create a simulation. It presents most of the important choices that go into a simulation and includes cross-references to actual commands. It also presents the most important conventions and the most common user errors. Chapter 3 describes how to execute the simulation, how to recognize errors, and how to continue the simulation further in time. Chapter 4 offers an abridged discussion of the MAGIC Command Language (MCL), which includes the use of constants, variables, and functions, and provides a basis for advanced data processing methods. Chapter 5 discusses the rules of syntax, or how to interpret arguments in the Manual to write commands that work.

Parts 2 through 7 contain the actual command descriptions. The Parts and Chapters are organized by function, thus making it easy to compare the alternatives available. Each command description includes command_name, function, syntax, arguments (definitions), description, restrictions, see also (cross-references), references (to literature), and examples. The Parts are as follows:

Part 2 (MCL commands)	— variables, functions, do-loops, etc.
Part 3 (time and space)	— spatial objects (points, lines, areas, volumes) and grids
Part 4 (spatial extensions)	— outer boundaries and transmission lines
Part 5 (properties)	— conductors, dielectrics, resistive properties, emission processes
Part 6 (algorithms)	— electromagnetic, current-density, and particle kinematics
Part 7 (output)	— an extensive selection of simulation output types and methods

1.3 SOFTWARE DESCRIPTION

MAGIC[1] is an electromagnetic particle-in-cell code, i.e., a finite-difference, time-domain code for simulating plasma physics processes, i.e., those processes that involve interactions between space charge and electromagnetic fields. Beginning from a specified initial state, the code simulates a physical process as it evolves in time. The full set of Maxwell's time-dependent equations is solved to obtain electromagnetic fields. Similarly, the complete Lorentz force equation is solved to obtain relativistic particle trajectories, and the continuity equation is solved to provide current and charge densities for Maxwell's equations. This approach, commonly referred to as electromagnetic particle-in-cell (PIC), provides self-consistence, i.e., interaction between charged particles and electromagnetic fields. In addition, the code has been provided with powerful algorithms to represent structural geometries, material properties, incoming and outgoing waves, particle emission processes, and so forth. As a result, the code is applicable to broad classes of plasma physics problems.

The MAGIC tool suite includes MAGIC2D, a two- and one-half dimensional code (2D fields and 3D particle kinematics), MAGIC3D, a fully three-dimensional counterpart to MAGIC2D, and POSTER[2], a general-purpose post-processor. It reads from the MAGIC standard database format. It can redisplay simulation graphics after a simulation is complete. Because of the portability of the database, POSTER can be used for data rendering on platforms other than the one on which the simulation is performed. All codes in the MAGIC tool suite are built on an application-independent software library. This includes the command language interpreter from which the user interface for each code is built and the DUMP utility which provides communication between the codes and the database. The use of this library speeds code development, but more importantly, it helps integrate the system and makes the codes similar to learn and use.

The MAGIC Tool Suite for Windows consists of four software applications. These are:

MUGMAN – This application is the MAGIC Tool Suite Manager. It contains a browser for the electronic copy of the MAGIC User's Manual. It also allows the user to launch the tiny versions of MAGIC2D and MAGIC3D, as well as POSTER. The MAGIC Tool Suite Manger includes a set of example input files for both 2D and 3D, and provides online copies of the MAGIC Training Seminars. Old input files may be edited using the MAGIC Tool Suite Manager and new ones constructed, saved, and run with the appropriate application. The MAGIC Tool Suite Manager includes button slots for a postscript viewer and a movie viewer/editor program.

MAGIC2DT – This application is the tiny copy of MAGIC2D, and it is freely distributed for demonstration of MAGIC2D's features. It contains all the features of the licensed versions of MAGIC2D. It is limited to tiny simulation size problems.

MAGIC3DT – This application is the tiny copy of MAGIC3D, and it is freely distributed for demonstration of MAGIC3D's features. It contains all the features of the licensed versions of MAGIC3D. It is limited to tiny simulation size problems.

POSTER – This application is the general purpose post-processor for the MAGIC database format. It is supplied with all versions of MAGIC.

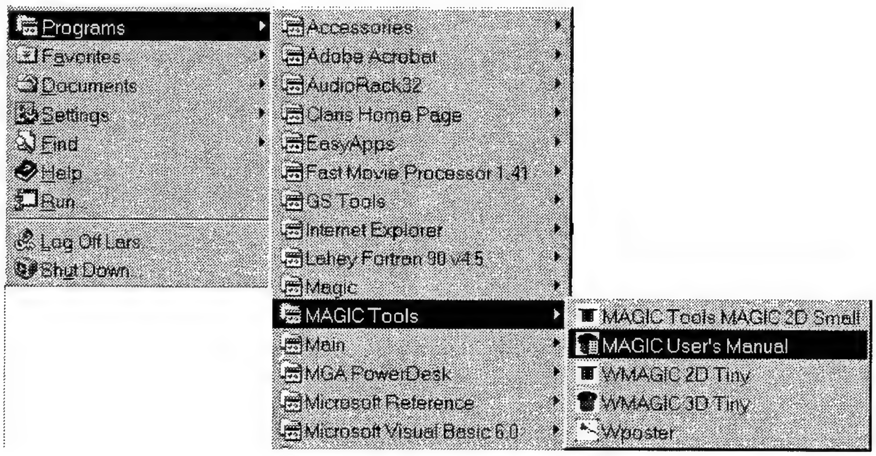
[1] B. Goplen, L. Ludeking, D. Smithe, and G. Warren, "User-Configurable MAGIC Code for Electromagnetic PIC Calculations," *Computer Physics Communications,* Vol. 87, Nos. 1 & 2, May 1995, pp. 54-86.
[2] L. Ludeking, Gary Warren, and Bruce Goplen, "POSTER User's Manual," Mission Research Corporation Report, MRC/WDC-R-245, August 1993.

1.4 STARTING THE MAGIC TOOL SUITE MANAGER

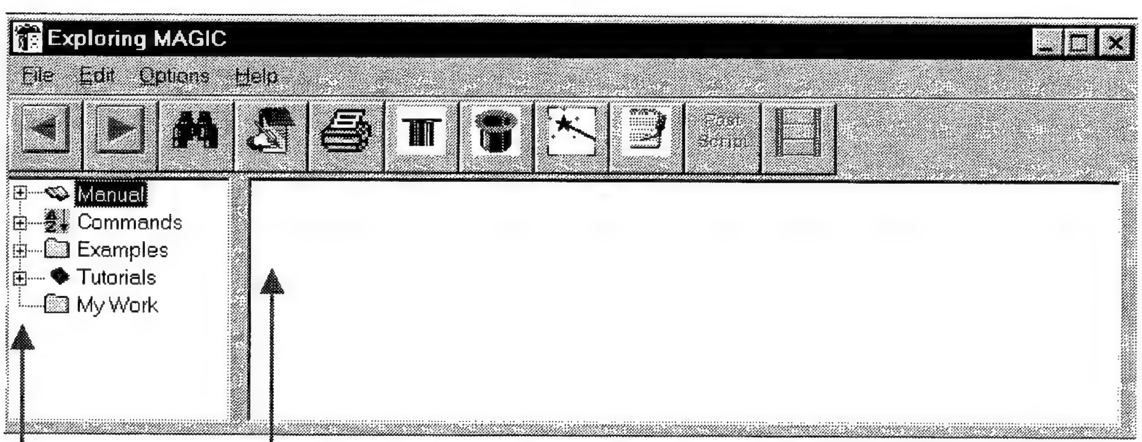
Launch the MAGIC Tool Suite Manager as follows:

⇛ Begin with the Windows START menu button,
⇛ Select the PROGRAMS menu,
⇛ Select the MAGIC Tools menu,
⇛ Click on MAGIC Tool Suite Manager.

The following figure illustrates what this will look like on your video screen.



Once the MAGIC Tool Suite Manager starts, it displays a window like the following one. This window may be resized just like any other Windows application to provide a bigger viewing area. Notice that it is split into two panes. The left pane is the navigation pane. It allows you to select from the items Manual, Commands, Examples, Tutorials, or My Work. The right pane is the text pane. This is where text is displayed once you have selected something to examine. In addition, the toolbar gives you control of other applications in the Tool Suite.



Text pane. Displays the text of the Manual, the Commands, the Example files, the Tutorials, and My Work files.

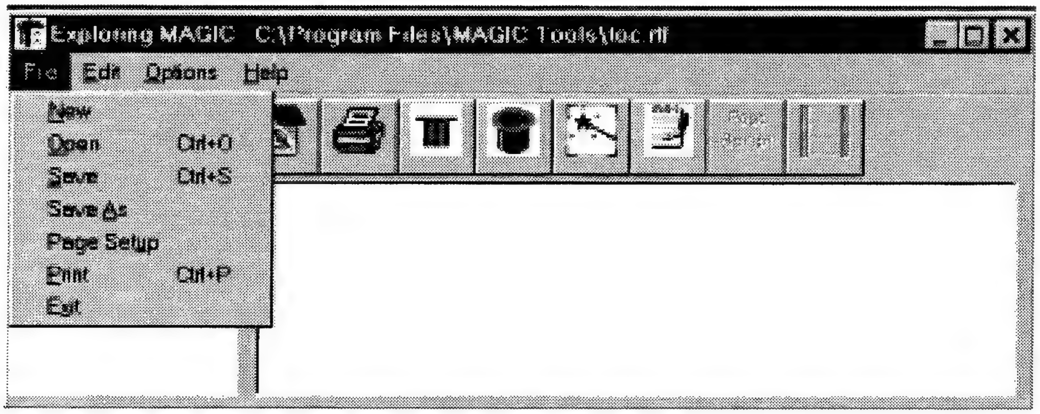
Navigation pane.

1.4.1 The Menus

There are four menus from which to select actions. Many of the menu actions are duplicated in the navigation pane and the toolbar. The following section briefly describes the menus.

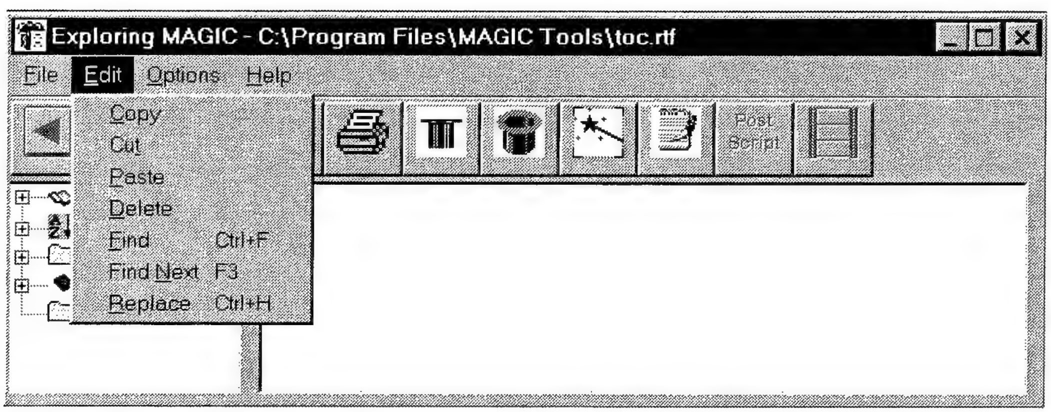
1.4.1.1 The FILE Menu

The FILE Menu looks and acts like most other file menus. For example, the Print option allows you to print whichever part of the user manual you are currently viewing in the text pane.



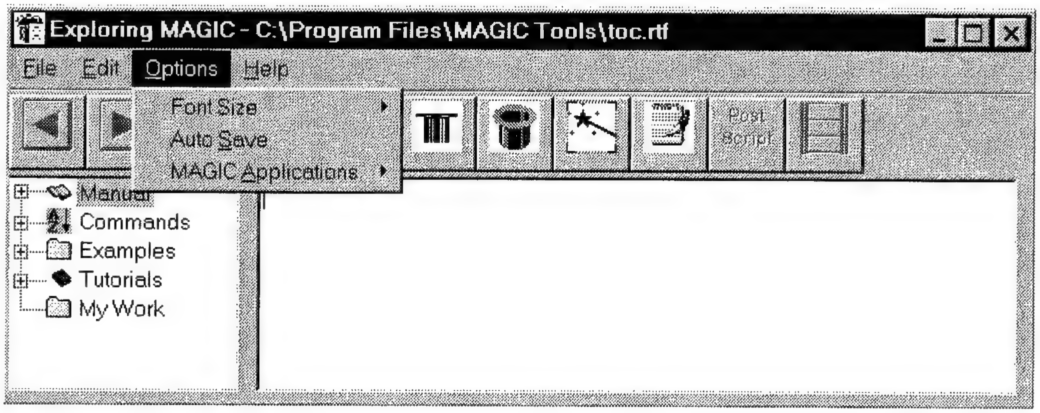
1.4.1.2 The EDIT Menu

The EDIT Menu looks and acts like most other edit menus. It permits you to copy text into the clipboard. For example, you might wish to copy and paste a section from an example into your input file.

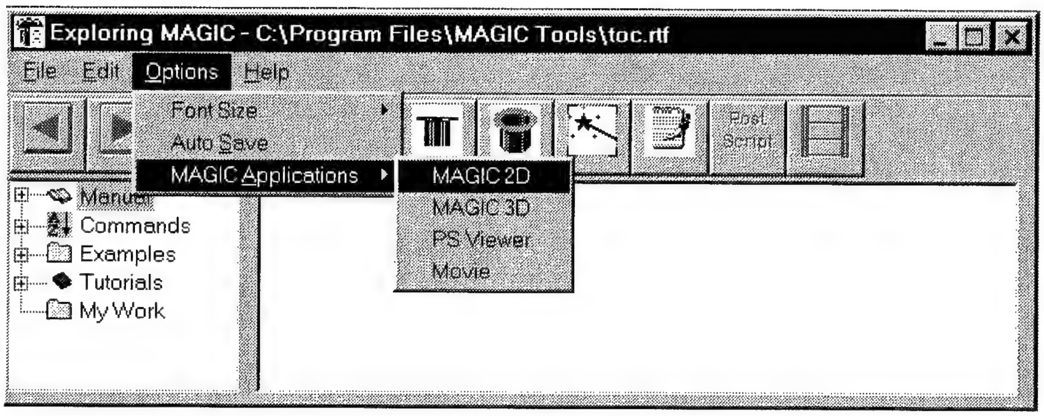


1.4.1.3 The OPTIONS Menu

The OPTIONS Menu allows you to configure the MAGIC Tool Suite Manager and its daughter applications to your requirements. By selecting the MAGIC Applications item you can alter the path and application that is attached to particular buttons in the toolbar.



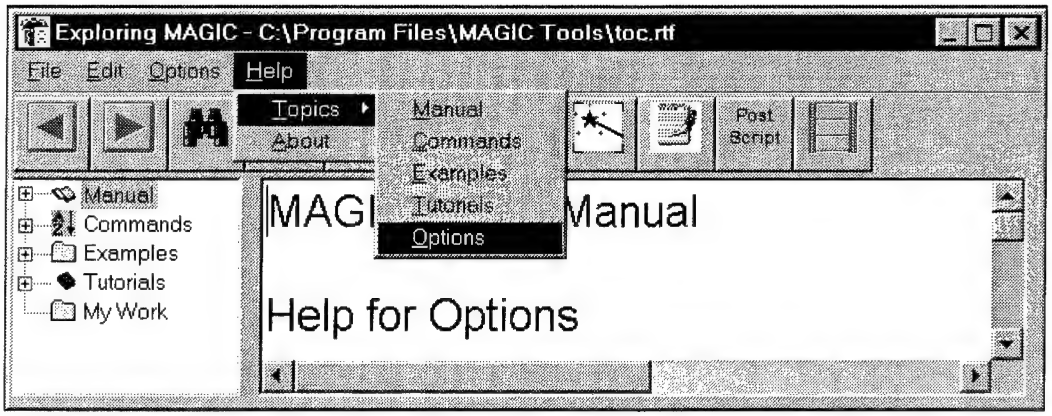
For example, select the MAGIC2D item in the popup menu.



A dialog box appears, giving you the opportunity to change the application path. For those who want to change the MAGIC2D application size from Tiny to another of the MAGIC2D sizes, enter the new path and the application name in the text box and then click okay.

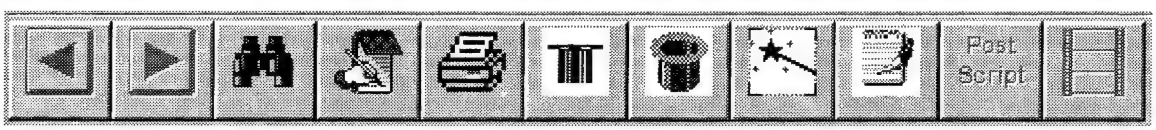
1.4.1.4 The HELP Menu

The HELP Menu gives you online assistance regarding the Manual, Commands, Examples, and Options. The help text is displayed in the text pane.



1.4.2 The Toolbar

The Toolbar contains several icons. This is the principal mechanism for launching the tool suite applications. Notice that the PSViewer and Movie buttons are grayed. You must use the Options menu to attach the desired application to these buttons.



Back Forward Search Notes Print Magic2D Magic3D Poster Wordpad PSViewer Movie

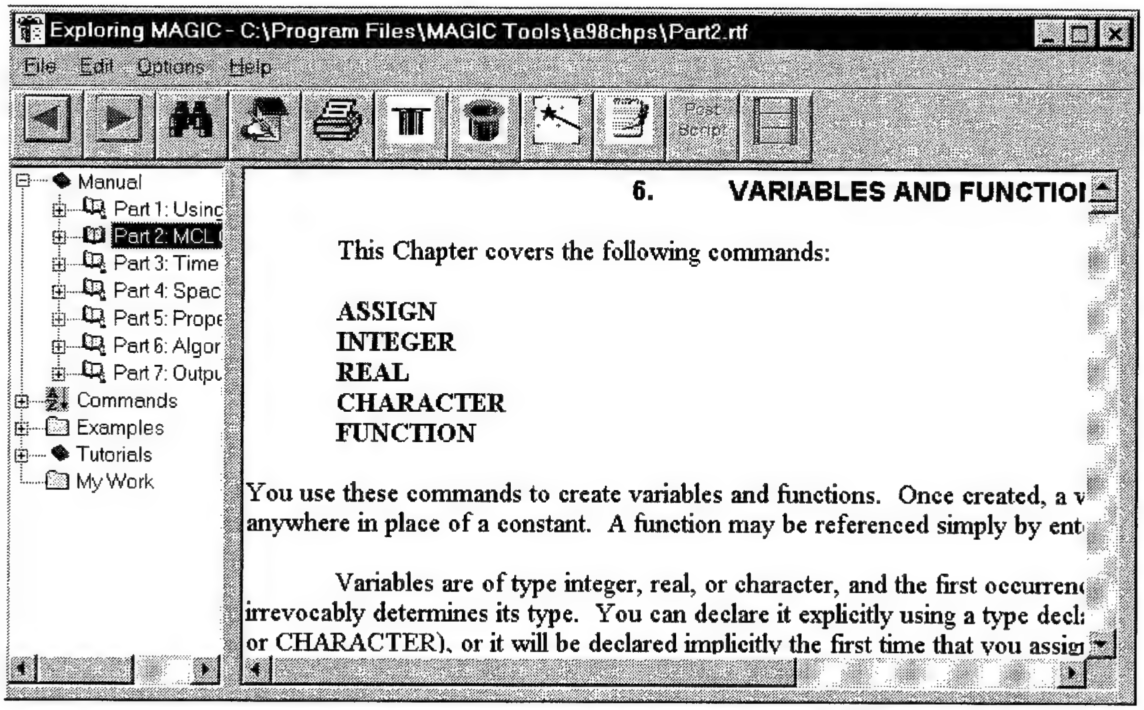
Button	Action
Back	This button is used to go back to the previous document. The arrow is gray when the option is inactive and red when active.
Forward	This button is used to go forward to the next document after using the back button. The center arrow is gray when the option is inactive and red when active.
Search	This button is used to search the text pane for a specific word or phrase.
Notes	This button starts the "my notes" editor. Use it to make notes in the manual.
Print	This button sends the contents of the text pane to the printer.
Magic2D	This button starts MAGIC2D.
Magic3D	This button starts MAGIC3D.
Poster	This button starts POSTER.
Wordpad	This button starts the wordpad text editor.
PSViewer	This button is for starting a post script viewer. You must first use the OPTIONS menu to specify a path and an application.
Movie	This button is for starting a movie editor/viewer application. You must first use the OPTIONS menu to specify a path and application.

1.4.3 Browsing The Magic User's Manual

To browse the MAGIC User's Manual use your mouse to select MANUAL in the navigator pane. Notice that the manual includes Parts 1 through 7. They are:

- Part 1 – Using MAGIC
- Part 2 – MCL Commands
- Part 3 – Time and Space
- Part4 – Spatial Extensions
- Part 5 – Properties
- Part 6 – Algorithms
- Part 7 – Output

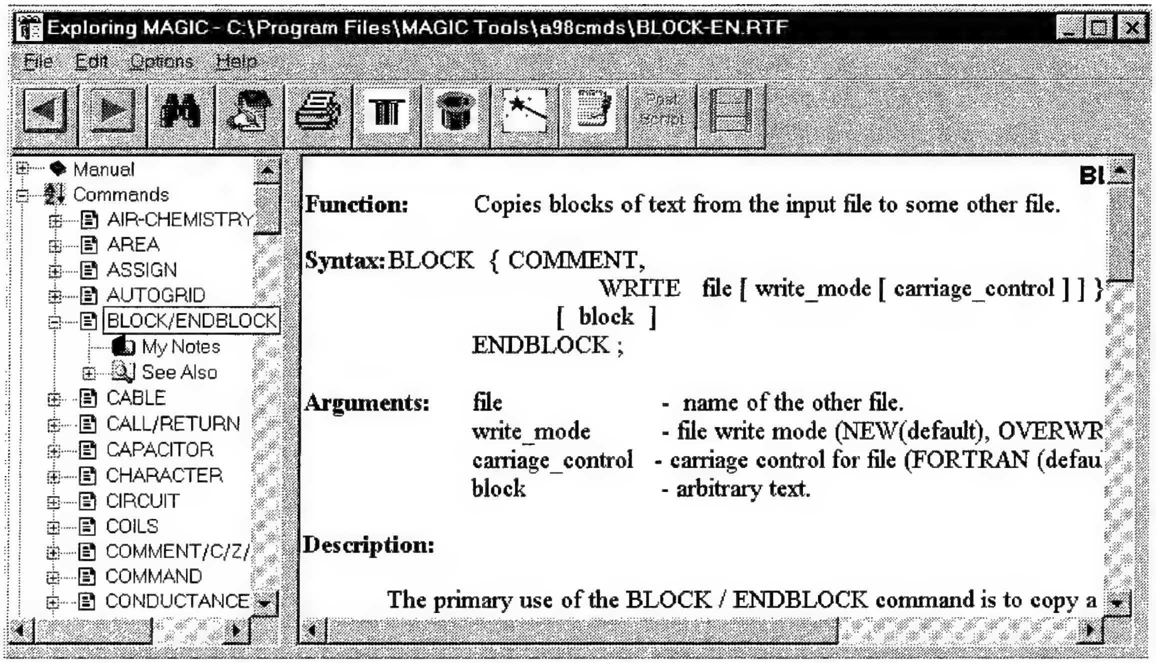
Select the part of the manual you wish to read. The manual text will appear in the text pane. The scrollbars allow you to move around in the section of the manual that you have selected.



1.4.4 Looking Up Individual Commands

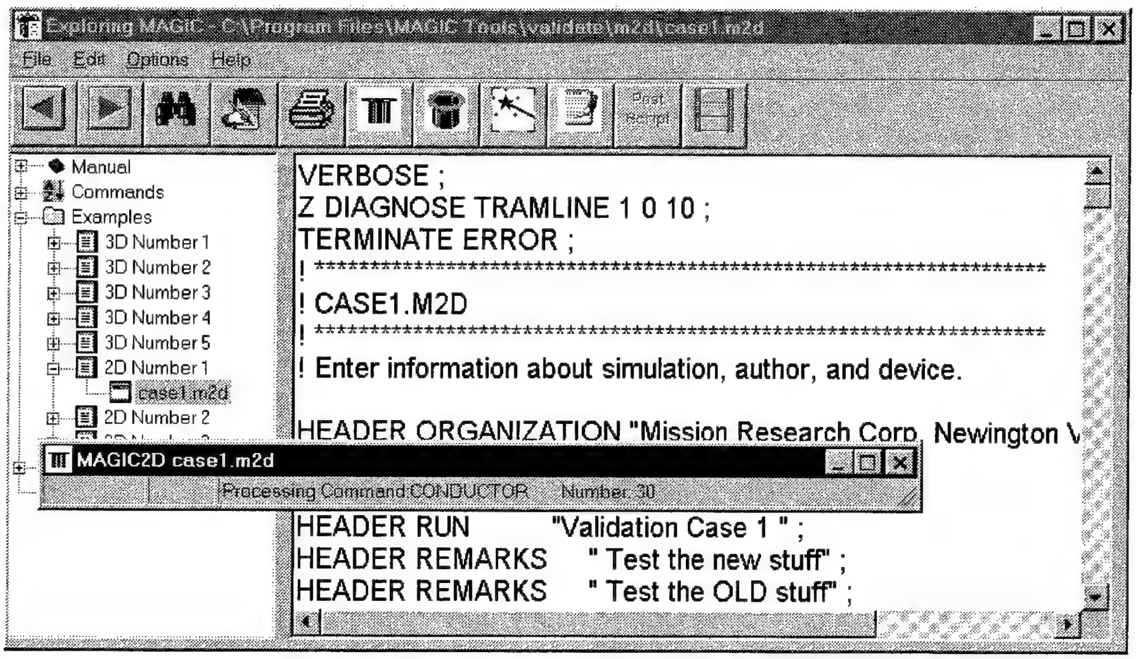
To look up an individual command in the MAGIC User's Manual use your mouse to click on the Commands item in the navigation pane. The display will expand one level and list the commands alphabetically. Use the central vertical scrollbar to position your mouse at the desired command. Using your mouse, click on the command you wish to browse. The command text will appear in the text pane. Double clicking on the command name will expand a sub-item list for the command, as illustrated in the following figure. The sub-item list will include an item called, "My Notes." By clicking on My Notes with the mouse, you will open a text window in which you can add your personal notes and comments about the command. You may also see an item called "See Also," which provides a short cut to related commands. Clicking on See Also will list the related commands. Clicking on one of these commands will take you to that command. You can return to your original position by clicking on the back arrow button in the menu bar. In some cases, there may be an additional item in the list. If

an example input file is available, it will appear in the list as the file name. Double clicking on the file name with the mouse will start either MAGIC2D or MAGIC3D, whichever is appropriate for running the example. All example input files also appear under the Examples list in the MAGIC explore pane.



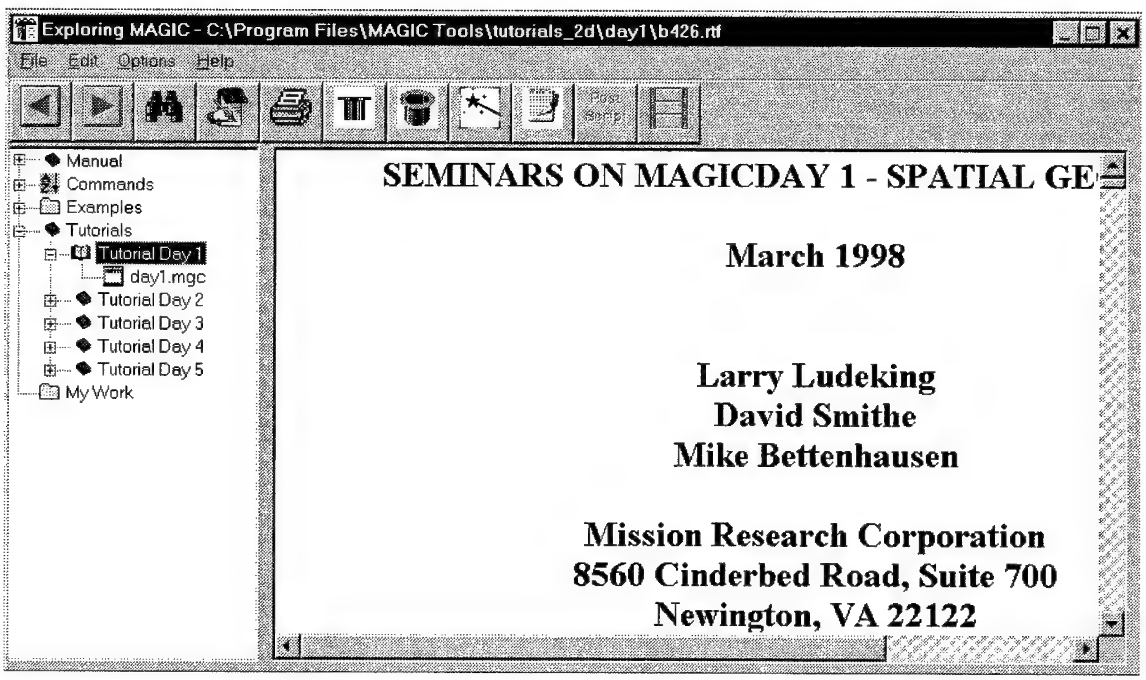
1.4.5 Exercising The Magic Examples

The MAGIC Tool Suite Manager includes several example input files. These files execute various features of MAGIC 2D and MAGIC3D. All of the examples are designed to run in a few minutes on a high end Pentium II platform. Using your mouse double click on the Examples item in the navigation pane. You will see a list of several examples. Using your mouse, single click on one of the examples. The text pane will give you a brief description of what the input file is modeling. Please read this carefully before running the example. In addition, the navigation pane will expand and list the name of the example input file. A single click on the input file name will cause the text of the input file to be displayed in the MUGMAN text pane. A double click on the example file name will start MAGIC2D/MAGIC3D and run the example file. Once the MAGIC example simulation has been started, use your mouse to ensure that the MAGIC application is the foreground Windows application. If the example simulation pauses for user intervention, it will display a message "Pause On" and "Waiting" in the lower left status display of MAGIC. Using your keyboard, press the Enter key to cause the simulation to proceed. If there are several graphics outputs to be displayed, you may need to do this several times. Should you wish to terminate the simulation before it is complete, press the ESCAPE key (followed by the Enter key if the 'Waiting' message is displayed).



1.4.6 Reviewing The Tutorials

The MAGIC Tool Suite Manager includes copies of the MAGIC training tutorials and the resulting training files. Using the navigation pane, select the tutorial of interest and review the material displayed in the text pane.



This page is intentionally left blank.

2. CREATING THE SIMULATION INPUT FILE

This Chapter explains how to create a simulation-input file. It includes a basic description of electromagnetic PIC, a command checklist, the most important conventions, and finally, the most common errors. Command names and keywords will be written in upper case to allow easy identification.

2.1 BASIC CONCEPTS

MAGIC simulates the interaction between charged particles and electromagnetic fields as they evolve in time and space from some defined initial configuration. The numerical calculation uses the finite-difference method. Time and three-dimensional space are divided into finite grids. From some known initial state, time is advanced by adding a single time step. At each new value of time, Maxwell's equations are solved throughout space to advance the electromagnetic fields in time. Using these new fields, the Lorentz equation is solved to advance the momenta and coordinates of all charged particles in the simulation. The continuity equation is solved to map charge and current densities onto the grid, which are then used as sources for Maxwell's equations on the next time step. This provides self-consistent interaction between the fields and particles.

Within this basic framework, MAGIC offers a selection of coordinate systems, grids, spatial objects, outer boundaries, material properties, emission processes, numerical algorithms, output, etc. This selection exists for two reasons, first, to cover a wide range of physical phenomena and second, to provide a high level of simulation fidelity. There are important attributes associated with the choices. For example, one electromagnetic algorithm may have superior stability, while another is more accurate. No single selection is best for all applications, and creating a simulation becomes largely a matter of defining the best tradeoffs between the choices. Defaults simplify selections for the novice.

The serious user will encounter cost limitations in his quest for simulation fidelity. A reasonable objective might be to achieve the highest quality consistent with time and cost constraints. To aid in this, a cost algorithm is essential. This algorithm, which can be developed for any platform, is best arrived at by empirical fits to actual simulation cost data. It is simply an equation which expresses time (or money) costs as a linear function of simulation parameters such as cell number, average particle number, and time steps. This cost algorithm will assist the user in making the tradeoffs essential to the necessary compromise.

2.2 COMMAND CHECKLIST

This checklist covers commands roughly in the order found in Parts 2 – 7. You will typically need commands from all six Parts, and it is good practice to enter the commands in the order listed below. There are other common-sense order requirements; e.g., a function must be defined before it can be used, and so forth.

Identification — You may provide background information to uniquely identify the simulation (HEADER, Ch. 20). This background information will be reproduced on virtually all simulation output. The default is blank.

Coordinate system — You must select a coordinate system (SYSTEM, Ch.10). The choices are Cartesian, cylindrical, polar, and spherical. The selection will fix the identification of generalized coordinates (e.g., x1, x2, x3) with physical (e.g., x, y, z) coordinates. (See Important Conventions, which follows.)

Variables — You may use variables as data entries anywhere in place of constants. However, you must assign values to such variables before they are used (ASSIGN, Ch. 6).

Spatial objects — You may create arbitrary spatial objects consisting of points, lines, areas, and volumes of diverse shapes (POINT, LINE, AREA, VOLUME, Ch. 10). The only attribute of spatial objects is geometric. You must use other commands to assign physical properties to such objects or to use them for some other purpose.

Spatial grid — You must construct a grid in two or three spatial coordinates, using either automatic or manual gridding (AUTOGRID or GRID, Ch. 11). To use automatic gridding, you must mark at least some of the spatial objects (MARK, Ch. 11). To use manual gridding, you must specify the grid origin (GRID ORIGIN, Ch. 11) followed by arbitrary regions of grid. The spatial grid is the primary determinant of accuracy (the time step is generally unimportant to accuracy), and all spatial phenomena of interest must be appropriately resolved.

Time — You must specify the time period to be covered by the simulation (DURATION, Ch. 11).

Field algorithm — You may replace the default electromagnetic field algorithm (MAXWELL, Ch. 17) and specify the time step (TIME_STEP, Ch. 17). The default field algorithms are suitable for mildly relativistic particle simulations, e.g., γ up to 1.5. The time step, spatial grid, and algorithm are the primary determinants of numerical stability. If the time step exceeds the Courant criterion, unambiguous catastrophic failure results. Highly relativistic particle simulations produce more noise, and numerical damping of this noise is another important property which depends on the choice of electromagnetic algorithm.

Physical properties — You may assign physical properties to spatial objects. Physical properties such as infinite conductivity, permittivity, etc. (CONDUCTOR, DIELECTRIC, etc., Ch. 14) can be assigned only to spatial areas (2D simulations) or volumes (3D simulations). Unique (fine-scale) structural properties, such as a strut, can be assigned only to spatial lines (STRUT, etc., Ch. 15). You cannot assign a physical property to a point.

Outer boundaries — You may assign outer boundary properties to spatial objects on the simulation perimeter. The perimeter shape and location is arbitrary (see Important Conventions, below). Most outer boundaries (SYMMETRY, PORT, OUTGOING, etc., Ch. 12) can be applied only to spatial lines (2D simulations) or areas (3D simulations). One exception (FREESPACE, Ch. 12) applies only to spatial areas (2D) or volumes (3D).

Initial conditions — Most simulations start from trivial initial conditions without fields or particles. However, you can supply non-trivial initial conditions explicitly by creating initial fields (PRESET, Ch. 19) and particles (POPULATE, Ch. 19).

Emission Processes — You may assign particle emission processes to spatial objects (EMIT, Ch. 16). However, you must first specify the process and parameters (EMISSION THERMIONIC, etc., Ch. 16).

Particle algorithms — You can alter the kinematics time step in the Lorentz algorithm (LORENTZ, Ch. 18) to help save time in non-relativistic simulations. You can also enter the manner in which the continuity equation is enforced in order to combat the growth of noise in high-density simulations (CONTINUITY, Ch. 19).

Output — You must explicitly specify any output required (various commands, Ch. 20–25). A timer is used to trigger output at different times during the simulation (TIMER, Ch. 11).

Finally — You must terminate the input file to execute the simulation (START, Ch. 9) or to stop processing (STOP, Ch. 9).

The flowchart below illustrates the order in which to create a MAGIC simulation. This order is designed to allow MAGIC to self-check many conditions that are necessary for simulation fidelity. It is good practice to follow this outline in all your simulations.

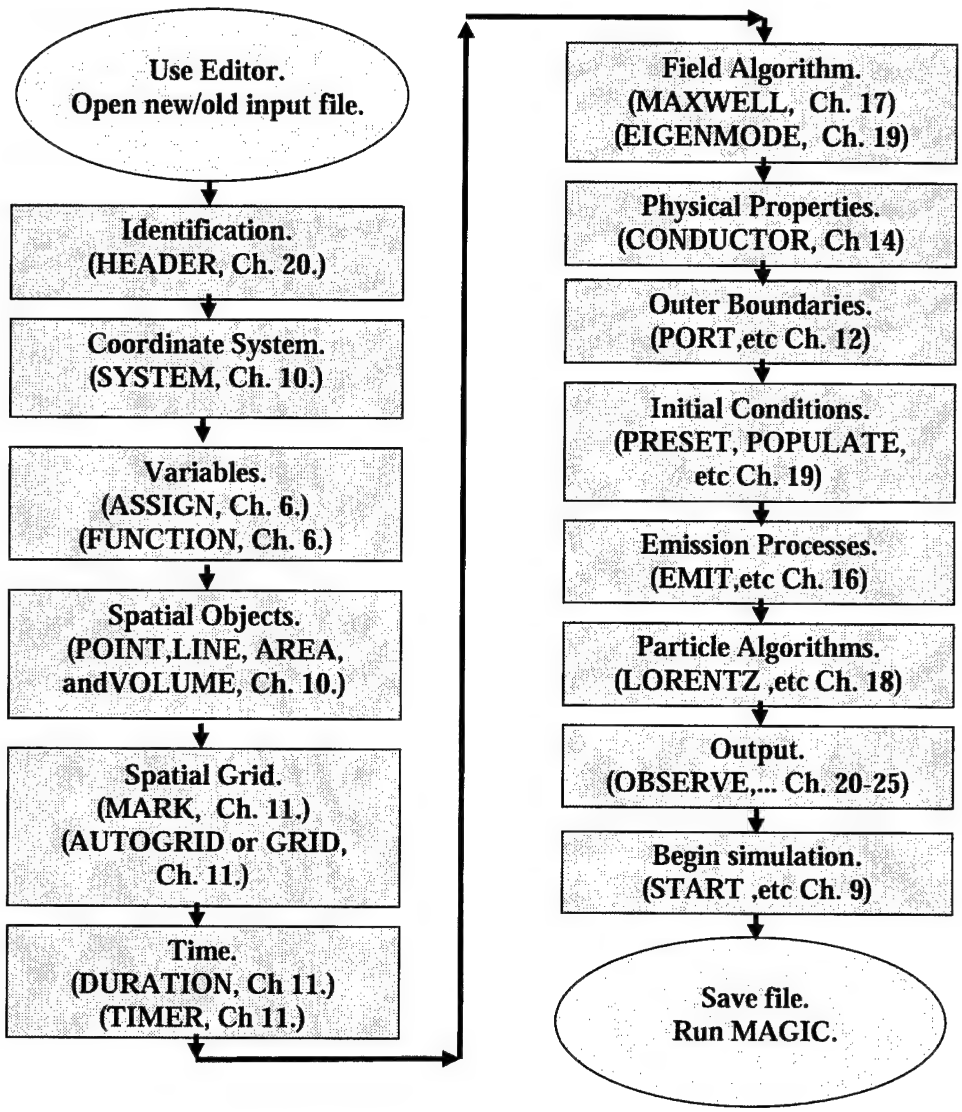


Figure 2-1. The COMMAND CHECKLIST (Good order in MAGIC file.)

A simple example of a MAGIC3D simulation file is included in the following table. This example follows the command ordering suggested.

Table 2-1. Simple example of a MAGIC3D simulation file.

```
TERMINATE ERROR;
HEADER  ORGANIZATION  "my organization" ;
HEADER  AUTHOR  "name" ;
HEADER  DEVICE  "Rectangular Cavity" ;
GRAPHICS PAUSE ;
SYSTEM CARTESIAN ;
Dx = 2.0cm ;
Dy = 1.5cm ;
Dz = 2.0cm ;
VOLUME BLOCK CONFORMAL -10CM,-15CM,-20CM +10CM,+15CM,+20CM ;
MARK BLOCK X1 SIZE DX ;
MARK BLOCK X2 SIZE DY ;
MARK BLOCK X3 SIZE DZ ;
VOLUME HOLLOW CONFORMAL -8CM,-13.5CM,-18CM +8CM,+13.5CM,18CM ;
MARK HOLLOW X1 SIZE DX ;
MARK HOLLOW X2 SIZE DY ;
MARK HOLLOW X3 SIZE DZ ;
AUTOGRID ;
CONDUCTOR BLOCK ;
VOID HOLLOW ;
EIGENMODE ;
DISPLAY_3D ;
CONTOUR FIELD E1 OSYS$MIDPLANE1 TSYS$EIGEN SHADE ;
CONTOUR FIELD E1 OSYS$MIDPLANE1 TSYS$EIGENMODE SHADE ;
CONTOUR FIELD E2 OSYS$MIDPLANE1 TSYS$EIGENMODE SHADE ;
CONTOUR FIELD E3 OSYS$MIDPLANE1 TSYS$EIGENMODE SHADE ;
VECTOR FIELD E2 E3 OSYS$MIDPLANE1 TSYS$EIGENMODE ;
VECTOR FIELD B2 B3 OSYS$MIDPLANE1 TSYS$EIGENMODE ;
START ;
STOP ;
```

2.3 IMPORTANT CONVENTIONS

Units — Input and output for MAGIC uses the MKSA system of units. The only exception is generalized particle momentum, which omits rest mass from the definition. In general, the correct physical units are stated in the command argument definitions and in printed and plotted output. You may enter constants using other units (ASSIGN, Ch. 6) and they will automatically be converted to MKSA values as they are processed.

Generalized coordinates — The choice of coordinate system determines not only the system, but also the identification with generalized coordinates (x1, x2, x3), as shown in the following table. This identification is important, because the command syntax uses the generalized coordinate notation.

System	(x_1, x_2, x_3)
cartesian	(x, y, z)
cylindrical	(z, r, ϕ)
polar	(r, ϕ, z)
spherical (2d)	(r, θ, ϕ)
spherical (3d)	(θ, ϕ, r)

In 2D simulations, the third coordinate (x_3) is ignorable. This has several important ramifications. First, the four systems are unique in 2D, while in 3D the cylindrical and polar systems simply have a different mapping to (x1, x2, x3). In 2D it is not necessary to supply the third coordinate (GRID, Ch. 11). Two (not three) values are sufficient to define a point in x_1, x_2 space (POINT, Ch. 10). It is also unacceptable to use the VOLUME command in 2D, since no depth coordinate exists.

In 3D simulations, no coordinate is ignorable. Thus, spatial grids are required for all three coordinates, and it takes three values (not two) to define a point. In addition to points, lines, and areas (all of which can exist in 2D simulations), we also have objects with depth called volumes (VOLUMES, Ch. 10). By inspection of the table above, note that the cylindrical and polar systems are identical (in 3D) except for the order of the coordinates. Therefore, the cylindrical and polar systems are redundant in 3D, and the choice is a matter of convenience.

Contiguous perimeter — The outer perimeter of the simulation must be contiguous, i.e., it must be completely enclosed. The perimeter must consist only of objects with assigned properties as conductors or outer boundaries (such as symmetries, etc.). However, it can use any combination of these, and it can be of any shape; i.e., it does not need to follow the extremes of the grid. The critical issue is that the perimeter not contain any "holes."

2.4 COMMON ERRORS

Some of the most commonly experienced errors are listed below. The severity of the result can vary substantially. Some errors (e.g., stability) result in catastrophic failure and are easy to recognize, while others simply diminish accuracy and are hard to recognize.

Failure to read the LOG file — Execution of MAGIC always results in a LOG file which lists any error messages (flagged with ***). Since not all errors terminate execution, searching the LOG file for errors is essential to assure simulation integrity.

Keyword duplication — Variable names which duplicate command names or previously defined function names can cause serious errors. In assigning variables, it is best to choose unusual variable names to minimize the possibility of duplication.

Contiguous perimeter violation — The outer perimeter of the active simulation region must be contiguous, i.e., the simulation must be completely enclosed by suitable boundaries. The critical issue is that the perimeter not contain any "holes," which could give wholly invalid results without necessarily producing an error signature. The code does not check this perimeter; it is entirely the responsibility of the user. However, in 2D simulations, you can use the DISPLAY PERIMETER command to inspect the perimeter visually (DISPLAY, Ch. 24).

Courant violations — There are several limitations on the size of the time step. The electromagnetic (Maxwell) algorithm fails catastrophically at 100% of Courant. You will know if this happens, since the exponential growth usually causes a system overflow. In certain coordinate systems and grids, failure may occur as low as 82% of Courant. Also, particles which travel more than one cell size in one time step will decouple in particle integer and coordinate space. This can result in a non-physical acceleration and even disappearance, and you may not notice it. In some outer boundaries (PORT and OUTGOING, Ch. 12), the time step must be less than the cell width divided by the phase velocity. These violations are more common where the choice of an implicit electromagnetic algorithm allows a larger time step. Some violations (but not all) are detected by the code and produce error messages.

Resolution errors — Inaccuracy can result if the cell-to-cell variation in any axis exceeds 25%, or the aspect ratio of the unit cell exceeds five to one. In addition, physical phenomena of interest must be adequately resolved by the time step and the spatial grid.

Outer boundary location errors — Placing an outer boundary too near a structural singularity, e.g., a corner or protrusion, can result in artificial scattering and even instability. Generally, outer boundaries expect the outgoing wave to be well behaved, and they do not handle fringe fields well. The best approach is to place the outer boundary at the end of a uniform channel of some length, and to avoid abrupt discontinuities in structure near the boundary. Waves propagating in the radial direction through a boundary are particularly troublesome and should generally be avoided or treated with great care.

Failure to test — This error results from the construction of an extremely large or complicated simulation without testing any of the models and features employed. Because of nonlinearity, it is difficult to recognize even the most basic errors in a complex simulation. Therefore, it is good practice to test elements of the simulation under idealized conditions.

3. EXECUTING THE SIMULATION

MAGIC is designed for serial execution of an input file. First, create an input file using a text editor. The MAGIC software automatically recognizes certain file extensions as belonging to either MAGIC2D, MAGIC3D, or POSTER. For MAGIC2D, we recommend using "M2D," for MAGIC3D, we recommend that you use "M3D," and for POSTER, we recommend that you use "PST." More information on file extensions is presented later.

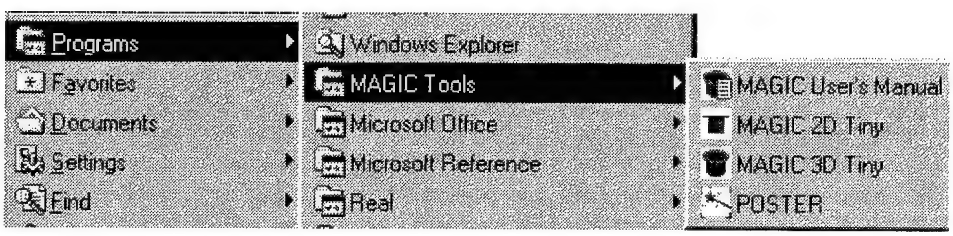
This Chapter describes how to start and interact with the simulation during execution, what to do with the output files, how to interpret error messages, and how to review the simulation results.

3.1 STARTING A MAGIC TOOL

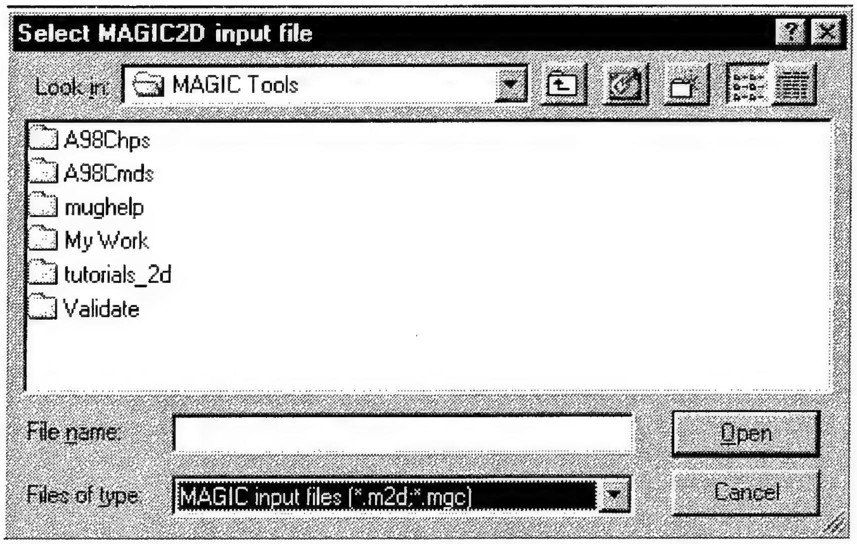
You can launch the applications in the MAGIC Tool Suite for Windows in the following manner:

⇛ Begin with the Windows START menu button.
⇛ Select the PROGRAMS menu,
⇛ Select the MAGIC Tools menu,
⇛ Click on the desired tool – MAGIC2D, MAGIC3D, or POSTER.

The following figure illustrates what this will look like on your video screen.

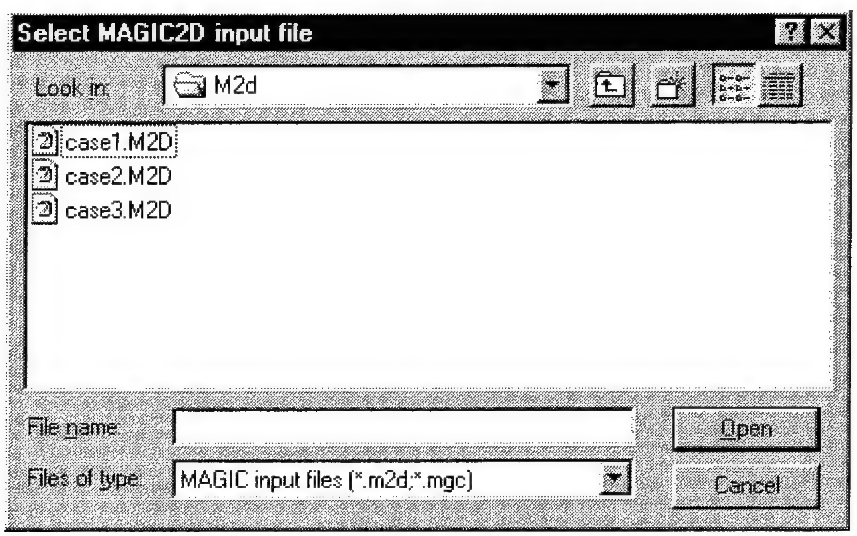


When the MAGIC tool starts, it will display a file requestor dialog window that looks like the illustration that follows.



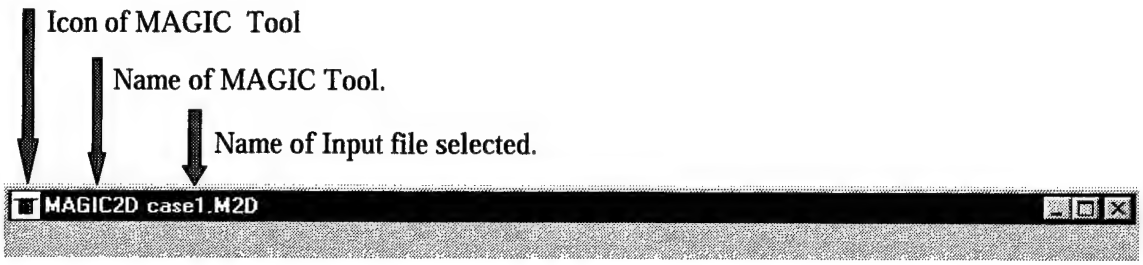
Use this dialog box to navigate to the desired folder that contains your input file. For MAGIC2D the default file extensions that are automatically displayed are M2D and MGC. For MAGIC3D the default file extensions that are automatically displayed are M3D and MGC. For POSTER the default file extensions that are automatically displayed are TOC and PST.

For example, suppose you select the VALIDATE folder and further select the M2D subfolder. Then the dialog display will look like the following.



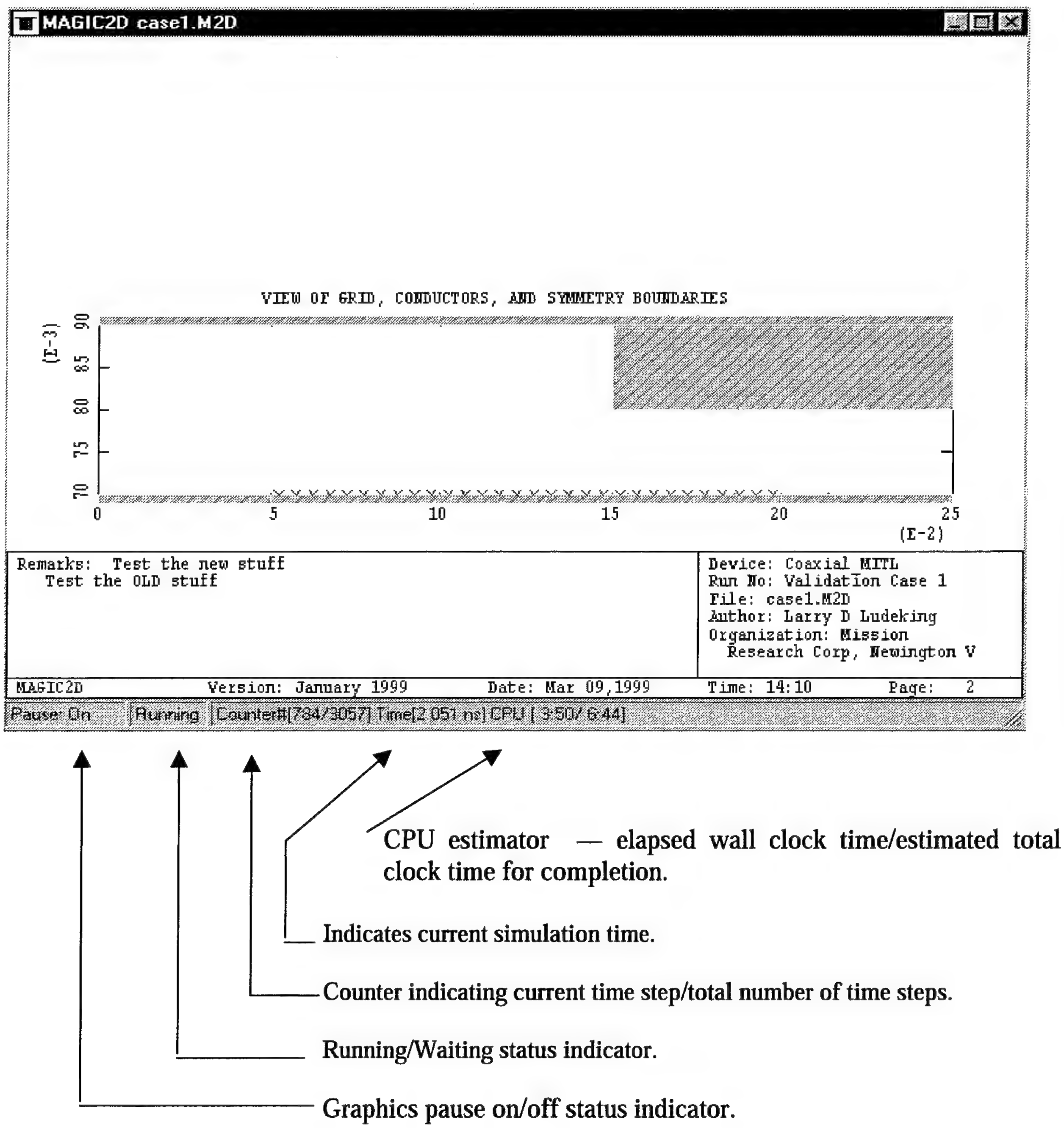
Notice that the text box displays files with the extension M2D, that are associated with the MAGIC2D tool. Using your mouse, select the desired file and then click on the Open button.

The next thing that will be displayed will look like the following.



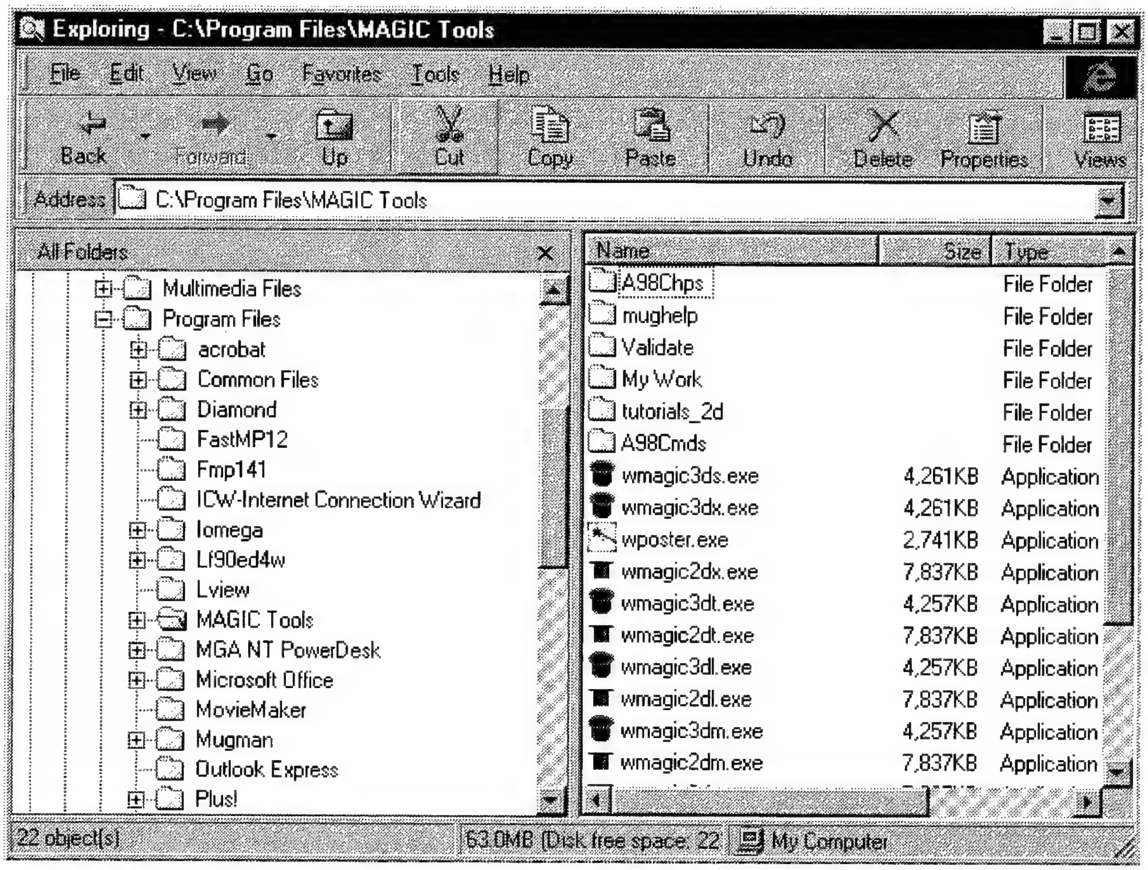
This bar appears when the MAGIC Tool starts to process the selected input file. The icon and name on the left end of the bar indicate which of the MAGIC Tools is being employed. In addition, the bar displays the name of the input file.

Once the input file has been processed, the MAGIC Tool will display a new window that resembles the following. After the first plot frame is displayed, the MAGIC Tool will pause (if enabled) before displaying any subsequent plot. Processing continues as soon as you press Return on your keyboard.

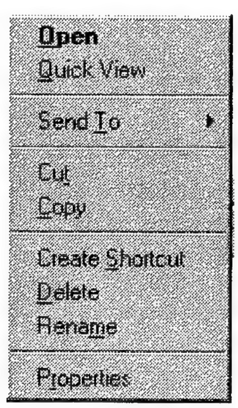


3.2 CREATING SHORTCUTS FOR MAGIC TOOLS

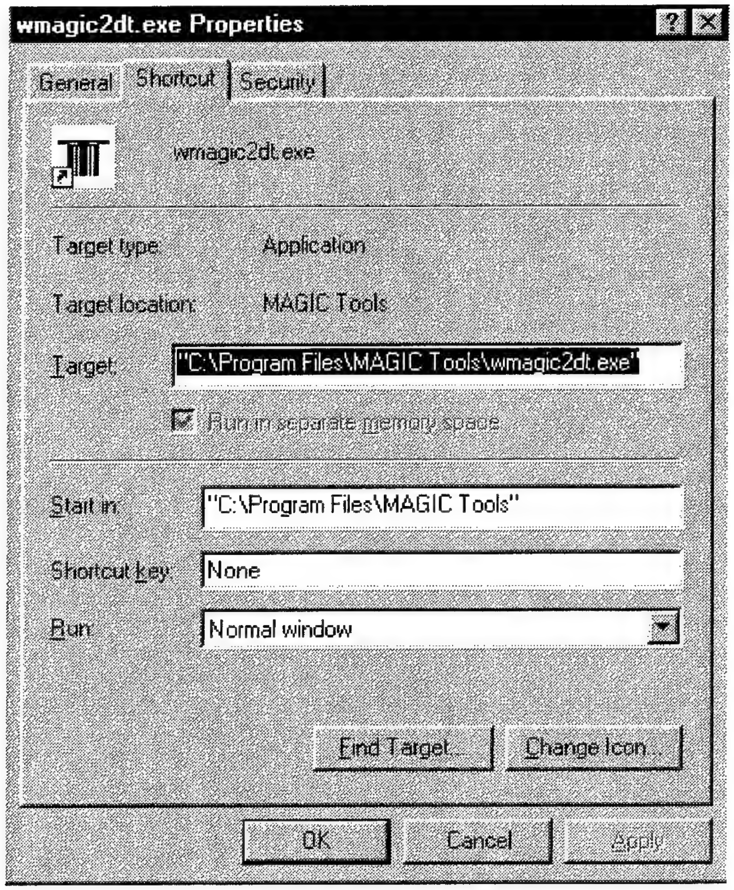
In using the MAGIC Tools you may have several working folders in which you keep different projects or sequences of simulations. You may find it convenient to create shortcuts in each of the working folders to the MAGIC Tool that you are commonly using. To create a shortcut, use Windows Explorer to look at the MAGIC Tools Folder.



Using your mouse, highlight the MAGIC Tool for which you want to create a shortcut. Right click the mouse button. A dialog box, like the following, should appear.



Use your mouse to highlight the "Create Shortcut" item. Click with the left mouse button. This will create a shortcut item in this folder. Cut and paste this shortcut item to your desired working directory. Using explorer and your mouse, highlight the shortcut item and right click on the property item in the menu. A dialog box should appear that looks like the following.



Notice the path in the "Start in:" text box. Remove the path information and leave the text box blank. Select Apply and click on the OK button. You can now start the MAGIC Tool in the working folder by double-clicking on it in Windows Explorer.

3.3 CONTROL KEYS

During execution, you may interact with the simulation through the keyboard (KEYBOARD, Ch. 9). For example, you can press the f6-key to send graphical output to the monitor (screen mode) or the Alt-f6-key to send it to the PS graphics file (printer mode). (The output mode can also be set by commands (GRAPHICS SCREEN and GRAPHICS PRINTER, Ch. 20).) Pressing other designated keys will override a command time-trigger (TIMER, Ch. 11) and create output precisely at the current time step. This allows you to examine results from a simulation in progress, rather than waiting for completion. The designated control keys and their functions are as follows:

Esc — writes all time history plots and terminates simulation being executed
f3 — turns PAUSE ON for graphics (screen mode only)
f4 — turns PAUSE OFF for graphics (screen mode only)
f5 — displays all time history plots up to current time step

f7 — creates a bit-map image file of the current plot display in a subfolder called BITMAP
f8 — displays all phase-space plots
f9 — displays all contour plots
f10 — displays all perspective plots
f11 — displays all range plots
f12 — displays all vector plots

In addition to the designated control keys, you can use an intrinsic function named LASTKEY in commands to provide customized interactive capability (FUNCTION, Ch. 6).

3.4 OUTPUT FILES

Depending upon the commands in the file, a single execution may produce as many as nine output files:

file_name.LOG — "log" of processed commands, errors, and printed output data.
file_name.SUM — "summary" of designated simulation variables and final values.
file_name.PS — "Postscript" printer file.
file_name.CGM — "CGM" graphics metafile.
file_name.TOC — "table-of-contents" to post-process FLD, GRD, and PAR files.
file_name.FLD — "field" data for post-processing.
file_name.GRD — "grid" and miscellaneous data for post-processing.
file_name.PAR — "particle" data for post-processing.
file_name.MVY — "movie contents" to play the movie files.

Each execution always produces a LOG file and a SUM file. The LOG file contains all processed commands; each data entry is written and interpreted on a separate line. Error messages are flagged with ***, to allow a quick search for errors. PRINT options in various commands may also write output to this file as the simulation progresses (STATISTICS, TABLE, etc., Ch. 21). Finally, basic simulation parameters, such as the number of cells, etc., are recorded. The SUM file records the final values of any simulation variables which have been specifically designated as simulation parameters (PARAMETER, Ch. 21).

The other files are produced only if triggered by certain commands in the input file. The PS printer file or CGM metafile is produced if a graphical output command is executed (various output commands, Ch. 22–24) while the system is in printer mode (see Control Keys, above). The type of printer file depends on the operating system. The PC generates Postscript commands, and the file is called PS. On UNIX workstations the file is called CGM. Workstations typically provide a post-processor (e.g., ctrans or gplot) which allows the user to view plots or to route them to a printer.

The TOC, FLD, GRD, and PAR files are produced only if post-processing is anticipated (DUMP, Ch. 25). Basically, any data that can be plotted directly can also be routed to one of these files. After the simulation, the files can be read into POSTER for processing and display. The MVY file is produced whenever the MOVIE option is included in an output command.

3.5 ERROR MESSAGES

The code tests individual commands as well as simulation consistency. When individual commands are read from the input file, the following errors can be detected.

invalid command_name	— misspelling or abbreviating the command_name
invalid key words	— misspelling (abbreviations are allowed within the command)
overused commands	— exceeding the maximum number of command repetitions

incorrect number of data entries	—	providing too few or too many data entries
incorrect data type	—	entering character data for integer or real data, or vice versa
out-of-range number	—	exceeding the platform word accuracy
undefined variable	—	trying to use a variable before assigning it a value
undefined function	—	trying to use a function before defining it

If no command errors are detected, the results are tested for the following errors:

spatial grid arrays (number of 1-D cells) exceeded
field arrays (number of 3D cells) exceeded
Courant stability criterion (time step) in electromagnetic algorithm exceeded
Courant stability criterion (time step) in boundary condition exceeded

If errors of any type are found, they will be flagged with *** in the LOG file, and the simulation will be terminated. You may also specify other error-related conditions for termination (TERMINATE, Ch. 9) e.g., TERMINATE ERROR is recommended for novice users. However, the code does not detect all errors. There are many desirable but unanticipated uses which are not safe, because they have never been tested. There are also common errors that are easy to warn against but difficult to test for. Chapter 2 includes a list of some of these. For now, the primary responsibility for error detection must rest with the user. Most realistic simulations contain compelling physical processes and nonlinear interactions and are so complex that it is often difficult to recognize errors. Therefore, we recommend that you build a simulation in parts and test each part extensively under idealized conditions.

3.6 REVIEWING THE SIMULATION GRAPHICS ON-SCREEN

The simplest way to review the simulation results, if the simulation has been run in GRAPHICS FILE (Ch. 20) mode, is to send the PS or CGM file to a printer. It is also possible to locate free or inexpensive shareware programs, e.g., ghostview (http://www.cs.wisc.edu/~ghost/), which will display the contents of these files on a computer screen, instead of on a printer.

There are considerable advantages to running a simulation in GRAPHICS WINDOW mode, though. It offers the ability to view the state of the simulation as it is running, permitting one to abandon the run if some aspect of the simulation is not as desired, or if a steady-state condition has been reached. When a simulation has been run in GRAPHICS WINDOW mode, there is no graphics file, which presents an apparent problem, when it becomes necessary to review the output after the simulation has completed. In this circumstance the DUMP (Ch. 25) command should be invoked. This will result in a TOC file being created, which is actually an input file for the POSTER program. Running the TOC file in POSTER will replay all of the graphics from the simulation on-screen. For reasons discussed in the next section, it is recommended that DUMP be used in virtually all simulations, including even those using GRAPHICS FILE, unless hindered by lack of file space.

3.7 MOVING GRAPHICS INTO OTHER PROGRAMS (Windows — PC)

The most straightforward way of moving graphics into other programs such as word processors and presentation managers is to utilize the DUMP capabilities and the TOC file. Simply replay the simulation graphics by running the TOC file in POSTER, and stop when the desired graphic is on screen. Press the Alt-Print Screen keys on the keyboard; this will place a bitmap image of the window to the clipboard. Then, Alt-Tab to another program window and paste.

It is often convenient to paste the bitmap into the Paint program first, in order to trim off undesired headers, modify colors, add labels, etc., before pasting the graphic into a third program in its final form. It is also sometimes convenient to edit the TOC file and comment out lines corresponding to unwanted graphics, so that they will not appear on the screen when POSTER is run. All users should eventually become familiar with the TOC file and how to use and edit it.

3.8 MOVING RAW DATA INTO OTHER PROGRAMS

There are times when a bitmap image of the simulation graphic is not acceptable. Examples include the need to display several curves in the same figure and the need to manipulate the data in some way before displaying it. This occurs most often for RANGE (Ch 23) and OBSERVE (Ch. 22) output. It is possible to move the raw data for these outputs into another program, typically a spreadsheet. Again, the key is to use the DUMP command, with ASCII format, and the TOC file. Each graphic output from MAGIC adds a line to the TOC file indicating the type of data, e.g., RANGE or OBSERVE, and the data's unique time and title identifiers. The GRD file contains the actual data for RANGE and OBSERVE, and a search of the GRD file for the desired title identifier will locate the data of interest, which appears in the form of two parallel columns. These columns of data can then be copied and pasted into another program, where they can be manipulated as suits the user.

3.9 VIEWING MOVIES (Windows — PC)

MAGIC offers simple movie capability, even for novice users. Viewing a simulation dynamically in movie format always improves the understanding of complex behavior so common in Maxwell-Lorentz physics; we strongly encourage all users to take advantage of this capability. Addition of the MOVIE keyword to any timer-based output command will activate the movie creation feature. The MVY file is actually an input file for MAGIC2D and MAGIC3D, in exactly the same way that the TOC file is an input file for POSTER. Running the MVY file in MAGIC2D or MAGIC3D shows a succession of movie clips, one for each command with a MOVIE keyword. As with the TOC file, the MVY file can be edited to remove undesired clips.

3.10 CONVERTING MOVIES TO AVI FORMAT (Windows — PC)

The built-in movie display format is intended more for everyday scientific use than for presentation. In order to produce a presentation quality movie, it is necessary to use additional video editing software. This software must convert the sequence of PCX files generated by MAGIC into a single AVI format movie file.

An easy-to-use shareware program called FASTMOVIE is distributed with the MAGIC Tool Suite by permission of the authors. It can also be found at the web site "http://www.gromada.com/fmp.html." This program is recommended to those with no prior video editing experience. If you find this program to be useful, we request that you abide by the shareware agreement, which includes payment of a minimal registration fee to the authors.

To make an AVI movie, first locate the sequence of PCX files comprising the movie frames. Normally MAGIC will create a subdirectory where the simulation input file is located and place the PCX files there. For example, a movie consisting of contour plots will be in a directory called CONTR001, and the first frame of the movie will be a file called FRM00001.PCX. To produce the AVI movie with the FASTMOVIE program, follow the following steps.

1) Choose "Add Input File" from the "Input" menu. This will bring up an open-file dialog window.

2) Select the PCX file representing the first frame of the movie.

3) Choose "Select File ..." from the "Output" menu. This will bring up a save-file dialog box. Navigate to the directory in which you want the AVI file to be created and make up a new name for your movie, e.g., MYMOVIE.AVI.

4) Once you have entered the AVI file name, an AVI Output Settings dialog box will appear. Use the default settings, which should include Codec compression, and select OK.

5) Choose "Begin Processing" from the "Output" menu. It may take several minutes to complete processing.

6) Choose the "Exit" menu. Using Windows Explorer, locate the AVI file and double click on it to run the movie.

Movie quality can be improved by increasing the frequency of the plots, e.g., reducing the increment used in the TIMER command referenced by the output command generating the movie. Movie quality can also be improved by using the appropriate AXIS options in the output command to prevent the axis limits and contour levels from changing during the movie.

4. MAGIC COMMAND LANGUAGE

Input data for the code is based upon the MAGIC Command Language (MCL). MCL is a sophisticated scripting language with many powerful features. This Chapter presents an abridged description of these features, which will be sufficient for the vast majority of applications. It emphasizes the importance of variables in creating input data. Users who require or desire to explore the more advanced features of MCL are advised to contact the authors.

A typical command in an input file consists of the command_name followed by arguments specific to that command, and the command is terminated with a semi-colon, as follows:

command_name argument argument... ;

This Chapter describes the capabilities of MCL to read data. Most of these capabilities can be exercised using commands found in Chapter 6 (Variables and Functions).

4.1 CHARACTER SET

The character set which MCL can read is based upon FORTRAN. In general, individual data entries may contain upper- and lower-case letters (MCL is case-insensitive),

A B C D E F G ... Z a b c d e f g ... z

digits,

0 1 2 3 4 5 6 7 8 9

and the eighteen special characters,

. % & < > ? _ ~ ` @ # ^ \ | { } [ ]

The other special characters are reserved and have special meaning within MCL. For example, the following seven characters are reserved for mathematical expression operations (the asterisk is also used for "wild-card" operations).

=	variable assignment and function definition
+	addition
-	subtraction
*	multiplication (*) or exponentiation (**)
/	division
( )	mathematical and dummy argument grouping (always in pairs)

The rest of the special characters are reserved for use as delimiters. For example, the command illustrated at the beginning of this Chapter is terminated (delimited) with a semi-colon, and the individual data entries are separated (delimited) with blanks (commas are equivalent). A complete list of the MCL delimiters follows.

;	command termination delimiter
blank	data entry delimiter (interchangeable with comma)

,	data entry delimiter (interchangeable with blank)
“”	character string delimiters (always in pairs)
‘ ’	variable substitution delimiters (always in pairs)
:	format delimiter (when used inside variable substitution delimiters)
!	comment delimiter
$	namelist delimiter

It is possible to re-define some, but not all, of the delimiter characters. For example, if an existing namelist file uses & as a delimiter instead of $, the MCL delimiter can simply be re-defined in order to read the data (Ch. 8).

4.2 CONSTANTS

MCL reads constants (literal data) of the following type: integer, real, and character string. The resulting word length and accuracy are platform-dependent.

An integer constant consists of one or more digits preceded by an optional sign (+ or -). If a real constant is entered for an integer, it will be truncated to integer value. Some integer constant examples are

+100 100 -9999

A real constant contains one or more digits with a decimal point located anywhere. The digits may be preceded by a sign (+ or -) and followed by an exponent letter (e or E) and more digits representing an exponent (which may also be signed). In addition, a real constant may be appended with an optional metric prefix (pico, nano, etc.) and a physical unit (inch, foot, etc.). (The constant, metric prefix, and physical unit may be separated by underscore (_) characters, if desired.) Table 4.1 lists all metric prefixes and physical units which are recognized by MCL, and the internal conversion factors. Upon reading appended units, MCL automatically converts the constant to the proper MKSA value. Some examples of identical real constants are

0.000001 1.0e-6 1micron 1_micro_meter 1micrometer 1e-3millim

Each of these data entries will be converted to 10^{-6} (meters) as it is read by MCL. (They are all equal.) Note that only those prefixes and units listed in Table 4.1 will be recognized by MCL.

A character string consists of a number of character constants (including embedded blanks and special characters) delimited by double quotation marks. The allowable length of a character string is platform-dependent. The string may contain any of the special characters (including delimiters and expression operators) except for the string delimiter itself. The case of any letters within the string will be preserved. If it contains a variable substitution delimiter, then the indicated substitution (see Variable Substitution, below) will be performed as the string is read. Typical uses of character strings include file names, titles, and axis labels. An example of a character string is

“Double-gap klystron design #07”

4.3 VARIABLES

Variables may be used in place of constants wherever a data entry is required. We advocate use of variables for a number of reasons. First, it allows input data to be self documenting. Although MCL supports an extensive commenting capability, variables provide meaning to the data directly. Second, fundamental constants and parameters can be defined once and used many places in an input file. It is much simpler (and safer) to define pi = 3.14159... and to use pi wherever it is needed than it is to repeat a long string of digits. Third, the

practice results in input files which are re-usable. By defining basic parameters (lengths, frequencies, etc.) as variables, one can quickly change data and run a new case without searching the entire file for constants.

MCL can read integer, real, and character variables. The type can be determined implicitly by the variable name itself. If the variable begins with the letters i – n, it is an integer variable. Otherwise, it is real. To circumvent this convention, or to define character variables, one simply uses explicit type-declaration commands (see Chapter 6). Examples of integer, real, and character variables are

```
Index = 999 ;   pi = 3.14158 ;   character alfa ;   alfa = “bravo” ;
```

Note that the variable alfa has been explicitly declared type character prior to its assignment; otherwise, “bravo” would be read as a real variable.

In addition to user-defined variables, MCL supports a number of system variables which are internally defined but can be accessed and used in the input file. System variables all contain the prefix ISYS$ (for integer variables) or SYS$ (for real variables).

4.4 VARIABLE SUBSTITUTION

Variable substitution is a very powerful MCL feature which provides a capability similar to that of arrays. The default delimiter for variable substitution is the single quotation mark. When MCL encounters these delimiters, it substitutes the current value of the enclosed variable. The following commands,

```
DO i = 1, 3 ;
      x'i' = i ;
ENDDO ;
```

will produce three real variables named x1, x2, and x3, with associated real values, 1, 2, and 3, respectively. Although this example uses an integer variable (i), real and character variables can also be substituted.

It is also possible to specify the substitution format. The formats available for the three data types are

type integer - In.m
type real - Fn.m, En.m, and Gn.m
type character - An.m

The standard FORTRAN rules apply for integer and real variables. For character variables, n is the number of characters to substitute, and m is the position in the existing character. As a simple example, if the statement in the block above is changed to read

```
x'i:I2.2' = i ;
```

then the resulting variable names would be x01, x02, and x03.

Variable substitution is also the only means of performing character string concatenation. In the following example, a substring is extracted and used to create a new character variable.

```
CHARACTER FULLNAME LASTNAME ;
FULLNAME = “David Jones” ;
LASTNAME = “'FULLNAME:A5.7'” ;
```

The character variable LASTNAME will contain "Jones."

4.5 FUNCTIONS

Many commands require specification of a function. MCL can read a general mathematical expression (using dummy arguments). The name of the function can then be entered into the command which requires the function. MCL supports the standard FORTRAN intrinsic mathematical functions. These, and any previously user-defined constants, variables, and functions can be included in the expression.

An example of a function command is

```
FUNCTION  V(t) = SIN ( omega * t )  ;
```

where omega has previously been defined as a real variable. Once this function has been entered, it can be applied wherever required simply with the data entry,

```
V
```

Note that this data entry does not include the parentheses or the dummy argument(s).

4.6 MATHEMATICAL EXPRESSIONS

Variables can be assigned values using complicated mathematical expressions in the FORTRAN syntax, including function references and variable substitution. The following example assigns a value to the variable "dt" using the MAX and SQRT functions and the system variable, SYS$DTIME.

```
dt = MAX( SYS$DTIME, SQRT(1_meter**2/vsqr) ) ;
```

4.7 IN-LINE MATHEMATICAL EXPRESSIONS

MCL commands accept in-line mathematical expressions, in place of variables or constants. This is primarily intended as a convenience, in order to prevent the profusion of needless variables for one-time use. There are tight restrictions on the use of in-line expressions, though. The following rules must be adhered to:

1) An in-line expression must start with either the "+" or "-" signs.
2) An in-line expression must not exceed 32 characters in length.
3) Variable substitution is not permitted in an in-line expression.

The following example uses two in-line expressions to create a line along the Y-axis from "y-dy" to "y+dy". Here "y" and "dy" are two previously assigned variables. Note the use of the "+" sign to indicate the start of an in-line expression.

```
LINE  line_name  CONFORMAL  0.,+y-dy  0.,+y+dy ;
```

4.8 GEOMETRY, MODEL, AND SPECIES NAMES

Many MCL commands define or use the names of geometry, the names of electromagnetic models, or the names of user-defined particle species. These names may be composed of the standard letters, digits, and 18 special characters. They cannot contain blanks. Their interpretation is case-insensitive. In this fashion they behave exactly as variables do. The following example creates a geometric point whose name is "origin" and creates a diagnostic to observe the E1 field at this point. Note the arbitrary case.

```
POINT Origin 0.,0.,0. ;
OBSERVE FIELD E1 ORIGIN ;
```

4.9 FILENAMES

Some MCL commands require filenames. The difficulty with filenames is that some platforms have case-sensitive file systems (UNIX) and some do not (DOS, Windows). In addition, some systems allow blanks in the filename, while others do not. The MAGIC program recognizes case-sensitivity of filenames and blanks whenever a file name is entered as a character string constant between double quotes. This is the safest way to use filenames. MAGIC also accepts filenames without quotes, if the name does not contain blanks. However, the case of the filename will be indeterminate, and MAGIC may or may not find an existing file of the specified name, depending on the operating system. In the first example below, data produced by running the Pandira magnetic field solver is saved in a file called "outSF7" and is used to preset the static magnetic field, B1ST, in MAGIC. The filename is enclosed in quotes to insure that the operating system searches for and opens the file with exactly the same case as specified. In the second example, MAGIC is being told to name the graphics printer file "Plot.ps," without regard to case. It is quite possible that the actual file will be created with the filename "PLOT.PS."

```
PRESET B1ST PANDIRA "outSF7" ;
GRAPHICS PRINTER Plots.ps ;
```

4.10 KEYWORDS AND OPTIONS

Many MCL commands require keywords in addition to requiring constants, variables, functions, mathematical expressions, and various names as arguments. The use of keywords is case-insensitive. Keywords may also be abbreviated. The use of keywords and options is discussed in more detail in Section 5.

Table 4.1. Prefixes and units with conversion values.

Prefixes

Prefix	Factor	Prefix	Factor	Prefix	Factor	Prefix	Factor
atto	10^{-18}	micro	10^{-6}	kilo	10^{+3}	peta	10^{+15}
femto	10^{-15}	milli	10^{-3}	mega	10^{+6}	exa	10^{+18}
pico	10^{-12}	centi	10^{-2}	giga	10^{+9}		
nano	10^{-9}	deci	10^{-1}	tera	10^{+12}		

Physical constants

Unit	Factor
sec	1
second	1
c	299792458
rad	1
radian	1
pi	3.14159265359
deg	pi/180
degree	pi/180

Length

Unit	Factor
mil	$2.54x10^{-5}$
mils	$2.54x10^{-5}$
inch	$2.54x10^{-2}$
foot	.0254x12
feet	.0254x12
micron	10^{-6}
mm	10^{-3}
cm	10^{-2}
m	1
meter	1
km	10^{+3}

Electromagnetic

Unit	Factor
A	1
amp	1
volt	1
kV	$1x10^{+3}$
tesla	1
gauss	$1x10^{-4}$
ohm	1
mho	1
henry	1
farad	1
coulomb	1
V	1
T	1
H	1
F	1

Energy

Unit	Factor
eV	$1.6021892x10^{-19}$
keV	$1.6021892x10^{-16}$
meV	$1.6021892x10^{-13}$
GeV	$1.6021892x10^{-10}$
joule	1
watt	1

Frequency

Unit	Factor
hertz	1
Hz	1
kHz	$1x10^{+3}$
MHz	$1x10^{+6}$
GHz	$1x10^{+9}$
THz	$1x10^{+12}$

5. INTERPRETING COMMAND SYNTAX

This Manual describes commands which use the following form of syntax:

COMMAND_NAME argument argument ... ;

This Chapter will explain how to interpret the syntax to write commands successfully. The basic rules are very simple.

- Wherever the command syntax contains an argument, you should replace the argument with a data entry. (Conceptually, this is equivalent to assigning a value to the argument.)
- Upper-case arguments indicate keywords, and the data entry should duplicate the argument exactly. (The command name is the pre-eminent example of a keyword.)
- Lower-case arguments should be replaced with data entries of your choice, using integer, real, or character data, which may be either constant or variable.
- Double quotation marks " " enclosing an argument require that a character string be entered. (The quotation marks must be included in the input data.)
- Parentheses ( ) embedded within an argument require that the name of a previously defined function be entered. (The parentheses and dummy arguments used to define the function are not entered.)
- Brackets [ ] enclose optional arguments. (The brackets themselves are not entered.)
- Braces { } enclose a list of arguments requiring selection; only one data entry is made in response. (The braces themselves are not entered.)
- An argument containing an asterisk * permits an asterisk "wild card" entry, which allows several options to be selected with a single data entry.
- Ellipsis ... indicate that more data of the preceding type may be entered.

For most users, adherence to these basic rules will suffice, and you may wish to skip the rest of this Chapter. Naturally, there are variations on the basic rules which allow greater flexibility in creating input data. The remainder of the Chapter will discuss some of these variations and present some examples.

5.1 UPPER-CASE ARGUMENTS

Any argument in upper case represents a keyword. This requires that the data entry duplicate the spelling of the argument exactly. Since MCL is case-insensitive, lower-case input data is acceptable, so long as the spelling is correct. All command names must be written out in full (not truncated). For example, if the command syntax reads

STOP ;

then

Stop ;

or any similar variation will be acceptable, but the name must be written out in full because STOP is a command name (the first argument in a command).

For upper-case arguments offering options or selection within the command, truncation of the data entry is acceptable. However, the data entry must duplicate enough of one argument to differentiate it from the other possibilities. For example, if the command syntax contains the arguments,

{ YES, NO }

then you can select YES by entering

Y

because it can be differentiated from the other possibility (N or NO) .

5.2 LOWER-CASE ARGUMENTS

A lower-case argument usually requires a specific type of input data. Generally, the type (integer, real, or character) will be indicated by the first letter of the argument or by the meaning which it conveys. (Functions and strings will be discussed separately.)

If the first letter of an argument is i – n, then an integer is usually required. (The integer can be either an integer constant or an integer variable.) For example, if the command syntax contains the argument,

index_limit

where both first letter and meaning suggest an integer, then a legal data entry would be the integer constant,

1000

Or, you can first place an implicit ASSIGN command (see Chapter 6) in the input file to create an integer variable,

LIMIT = 1000 ;

Then you can enter the integer variable (LIMIT) instead of the integer constant (1000). Note that a variable must be assigned a value before it can be used, i. e., the ASSIGN command must come first in the input file.

If the first letter of the argument is not i – n, then real data is usually required. (Again, real data can be either a constant or a variable.) For example, if the command syntax contains the argument,

angle

suggesting a real value, then the real constant,

3.14159

would provide a legal response. Equivalently, you can define a real variable and enter the variable instead of the constant.

The argument context may indicate that character data is required. (Again, the character data can be either a constant or a variable.) For example, if the command syntax contains the argument,

object_name

then it is clear that character (rather than real) input data is required. A legal response is the character constant,

Line_A

Equivalently, you can define a character variable to be equal to Line_A and then enter the character variable as input data.

5.3 STRING ARGUMENTS

Double quotation marks " " enclosing an argument require that a character string be entered. The entry must include the quotation marks as delimiters. An example is the HEADER command which encloses the argument, remarks, in double quotes,

HEADER REMARKS "remarks" ;

This is clearly a requirement for a character string which will provide background information on the simulation. A legal response would be

HEADER REMARKS "TWT - 1099, Mk II, 3/1/96" ;

Note that the double quotation marks must be included as part of the data entry. This character string with embedded blanks and special characters will be preserved exactly as it appears in the command. Other applications requiring character strings include the COMMENT command and certain file specifications.

5.4 FUNCTION ARGUMENTS

Parentheses ( ) embedded within an argument indicate that the name of a function be entered. Before you can enter this name, you must first create the function by placing a FUNCTION command (see Variables and Functions Chapter) in the input data file. For example, if the command syntax contains the argument,

voltage(time)

the embedded parentheses indicate that a function is needed. (From the argument itself, you might surmise that you are being asked for a voltage as a function of time.) Let us assume that you choose a sinusoidal dependence with an amplitude of 100 V and a frequency of 10 GHz. You place in the input file a FUNCTION command,

FUNCTION V(t) = 100. * SIN (2 * 3.14159 * 1E10 * t) ;

(A function expression will accept constants, variables, mathematical operators, intrinsic functions, and even functions defined in previous commands.) Now, you can simply enter the name of the function (V) in the original command. Note that the data entry is V and not V(t); the dummy argument and parentheses are not entered.

For certain arguments which call for a function, it may be acceptable to enter a constant or a variable instead of a function. (This is equivalent to defining a function equal to a constant, but more convenient.) Those cases where constant data is acceptable will be explicitly identified in the argument description.

5.5 OPTIONAL ARGUMENTS

Brackets [ ] are used to enclose optional argument(s). For example, if the command syntax contains the argument,

[FFT]

then legal data entries include either FFT or nothing (blank). If the brackets enclose a number of arguments, the same number of data entries (or none) must be entered. Note that the brackets themselves are not entered in the input data.

5.6 ARGUMENT SELECTION

Braces { } are used to enclose a list of arguments and to mandate a selection. In response, you must make one (and only one) data entry. For example, if the command syntax contains the arguments,

{ A, B, C }

then legal responses include only A, B, or C. No other (e. g., a blank) response is allowed, because you must select from the enclosed list. The braces themselves are not entered.

5.7 "WILD-CARD" ARGUMENTS

A "wild-card" argument (*) allows you to select more than one option with a single data entry. During processing, the asterisk is replaced with all possible variations allowed in the selection. If an argument within the bracket or braces contains an asterisk (*), then wild-card entry is allowed. For example, if a command syntax includes

{ TE, TM, * }

then legal entries include TE, TM, or *. The * entry provides the means to select both TE and TM. Note that wild-card entry is supported <u>only</u> where explicitly stated in the syntax.

5.8 REPEATED ARGUMENTS

Ellipsis ... indicate that more data of the type suggested by the preceding argument(s) may be entered. For example, if the command syntax contains

index ...

then several integers (but at least one) should be entered. In general, the argument grouping which precedes the ellipsis defines the scope of the data which is to be repeated. For example, if the command syntax contains the arguments,

x, y ...

then it is clear that one or more pairs of data should be entered.

6. VARIABLES AND FUNCTIONS

This Chapter covers the following commands:

ASSIGN
INTEGER
REAL
CHARACTER
FUNCTION

You use these commands to create variables and functions. Once created, a variable may be used as a data entry anywhere in place of a constant. A function may be referenced simply by entering its name.

Variables are of type integer, real, or character, and the first occurrence of a variable in the input file irrevocably determines its type. You can declare it explicitly using a type declaration command (INTEGER, REAL, or CHARACTER), or it will be declared implicitly the first time that you assign a value to the variable. Either way, once the type has been declared, it cannot be altered by any subsequent command. Trying to change the type explicitly will produce an error. Trying to change it implicitly may alter the assigned value or produce an error.

The ASSIGN command is used to assign a value to a variable (variable = value). The value may be changed at will, consistent with the original type declaration. ASSIGN provides implicit type declaration. If the variable name begins with i – n, it is type integer. Otherwise, it is type real. But if the assigned value is enclosed in double quotation marks, then the variable is type character. The ASSIGN command also supports pseudo-array (multi-variable) capabilities and the optional specification of physical units.

The FUNCTION command is used to create functions out of mathematical expressions or numerical data. Virtually any expression which is legal in FORTRAN can be entered with this command. In addition to constants, variables, and intrinsic mathematical functions, this command will accept any function previously entered in the input file. It will even accept time-history data and user instructions during the simulation, thus providing feedback loop capability. The FUNCTION command also supports true vector (multi-value) capability.

ASSIGN Command

Function: Assigns a value to a variable.

Syntax:

[ASSIGN] variable = expression, ... ;
[ASSIGN] variable = "string", ... ;
[ASSIGN] variable, ... ;

Arguments:

variable	-	name of integer, real, or character variable.
expression	-	arithmetic expression in the FORTRAN style containing constants, variables, intrinsic functions, user-specified functions, the mathematical operators add (+), subtract (-), multiply (*), divide (/), and exponentiate (**), and the Boolean operators (.GT.,.LT., .LE., .GE., .EQ., and .NE.).
string	-	a character string (with double-quotes), or the name of a previously defined character variable (without quotes).

Description:

The ASSIGN command assigns a constant value to a variable. It can also be used to display the present value of a previously assigned numeric variable. As indicated by the syntax brackets, entry of the keyword, ASSIGN, is optional. If it is omitted, we refer to this as an "implicit" ASSIGN command.

The ASSIGN command can be used with integer, real, and character variables. Unless the type is declared explicitly in a type declaration command, it is declared implicitly in the first ASSIGN command. The implicit rules involve the variable name and the assigned value. The first character in the variable name must be a letter. If the letter is i – n, the variable is type integer. If the letter is a – h or o – z, the variable is type real. But if the assigned value is enclosed by double quotes, the variable is type character. To override these rules, you must use one of the type declaration commands. Once the variable type is declared, it can never be changed. However, the value assigned to the variable can be changed at will, using more ASSIGN commands.

An integer or real variable is assigned the value resulting from evaluation of an expression. The expression may include constants, previously assigned variables, basic FORTRAN and Boolean operators, intrinsic mathematical functions, and user-defined functions (see FUNCTION command). The ASSIGN command converts the resulting value from real to integer and vice versa, as appropriate for the variable type. ASSIGN also allows specification of physical units, which may be appended directly onto the value or separated from it by an underscore character. (When physical units are included in any data entry, the numerical value will automatically be converted to the equivalent MKSA value.)

For a character variable, double quotes around a string are required to preserve letter case, blanks, and other special characters in the string. If the value (string) is another character variable, then the double quotes are not used; however, both variables must be previously declared to be type character. The length of the string is the total number of characters between the quotes, including blanks. The maximum allowable length is 132 characters.

The ASSIGN command also supports a pseudo-array (multi-variable) capability. As the syntax indicates, it is possible to enter more than one expression, but using only one variable name. What happens in this event is that MCL automatically creates additional variables by appending an underscore and increasing integers to the variable name (see Examples, below). Note that this is not a true array, since the variables all have different

names. The pseudo-array capability is available for all variable types. Commas are required to separate the multiple data entries (spaces cannot be used as delimiters in this command).

Once a variable has been assigned a value, the variable can be used as a data entry anywhere in place of a constant. There are some restrictions. For example, if the selected variable name duplicates a keyword, then it will be used in place of the keyword, usually with disastrous results. (FORTRAN behaves in a similar manner; e.g., if you define a variable named SIN, it replaces the intrinsic trigonometric function.) Also, real and integer variables should never be used as data entries where a character string is expected, and vice versa.

Restrictions:

1. A variable must be assigned a value before it is used.

2. Type declaration is implicit in an ASSIGN command, and it is explicit in type declaration commands. Type declaration (implicit or explicit) occurs wherever the variable name first appears in the input file.

3. The variable name should not duplicate any keyword in the input file. For example, DATA cannot be used for a variable name if the DATA option in a FUNCTION command is used.

4. There is no string concatenation operator; instead, variable substitution is used to assemble strings (see Examples, below).

5. The maximum number of elements in a pseudo-array assignment is 50.

See Also: **CHARACTER, Ch. 6**
INTEGER, Ch. 6
REAL, Ch. 6
FUNCTION, Ch. 6

Examples:

1. The following two ASSIGN commands are equivalent.

```
ASSIGN c = 2.993e8;     ! This is a conventional ASSIGN.
c = 2.993e8;            ! This one is an implicit ASSIGN command.
```

2. The following two commands with appended physical units are also equivalent. Both data entries will be converted to 0.1 (meters) as they are read.

```
x = 10cm ;       x = 10_cm ;
```

3. The following sequence of commands computes the relativistic momentum of an electron with a velocity of 10^6 m/sec.

```
V = 1.E6 ;
C = 2.998e8; RM = 9.1E-31;  ! Speed of light, rest mass.
Gamma = (1 - (V/C)**2) ** (-.5) ;
P = Gamma * RM * V ;
```

4. The following commands illustrate the pseudo-array (multi-variable) capability. The single command,

```
x  =  1.0,  4.0,  9.0  ;
```

will produce precisely the same results as the following six commands.

```
INTEGER x_dim ;   x_dim = 3 ;   x = 1.0 ;
x_1 = 1.0 ;  x_2 = 4.0 ;  x_3 = 9.0 ;
```

That is, both produce the same five variables and values. The integer variable x_dim is automatically created to record the number of subscripted variables created. Note that the base variable x (without underscore or integer) is also created and assigned the value of the first data entry. The same variables and assignments could also be produced explicitly using variable substitution, for example,

```
x = 1.0 ;
DO i = 1, 3 ;
      x_'i' = i ** 2 ;
ENDDO ;
```

5. The following example demonstrates type declaration and assignment for character variables.

```
Alfa = "Design # 4993" ; ! Quotes implicitly declares type character.
CHARACTER Beta ;         ! Type explicitly declares character.
Beta  =  Alfa  ;         ! Quotes are not required on Alfa.
```

Note that Alfa and Beta must both be declared type character before the ASSIGN command.

6. In the following example, concatenation of character strings is used to produce a file name for a CALL command.

```
CHARACTER FILENAME, PATH, FULLPATH ;
FILENAME = "Input.mgc" ;
PATH = "C:\Inputdir\" ;
FULLPATH = "'PATH''FILENAME'" ;
CALL FULLPATH ;
```

7. The ASSIGN command can also be used to display the present value of a previously assigned numeric variable. For example, the command

```
x_3 ;
```

will display the value nine (see Example 4). This capability does not exist for character variables.

INTEGER Command

Function: Declares a variable to be type integer.

Syntax: INTEGER variable, ... ;

Arguments: variable - name of a variable.

Description:

The INTEGER command explicitly declares a variable to be type integer. In an implicit declaration (see ASSIGN command), an integer variable must begin with a letter, i – n. Hence, the real purpose of the INTEGER command is to allow variables that begin with other letters to be type integer.

Any number of integer variables may be declared in a single command. If explicit declaration is used, it must precede the variable assignment and use. Once declared (implicitly or explicitly), the variable type can never be changed, but new values may be assigned to the variable, as convenient. The integer variable can be used in place of an integer constant wherever one is required as a data entry.

See Also: ASSIGN, Ch. 6
CHARACTER, Ch. 6
REAL, Ch. 6

Examples:

In this example, the variable X is declared to be type integer. In the absence of this explicit declaration, the variable would be real, and the integer constant (1) would be converted to a real constant (1.0).

```
INTEGER X ;
X = 1 ;
```

REAL Command

Function: Declares a variable to be type real.

Syntax: REAL variable, ... ;

Arguments: variable - name of a variable.

Description:

The REAL command explicitly declares a variable to be type real. In an implicit declaration (see ASSIGN command), a real variable must begin with a letter, a – h or o – z. Hence, the real purpose of the REAL command is to allow variables that begin with other letters to be type real.

Any number of real variables may be declared in a single command. If explicit declaration is used, it must precede the variable assignment and use. Once declared (implicitly or explicitly), the variable type can never be changed, but new values may be assigned to the variable, as convenient. The real variable can be used in place of a real constant wherever one is required as a data entry.

See Also: **ASSIGN, Ch. 6**
CHARACTER, Ch. 6
INTEGER, Ch. 6

Examples:

In this example, the variable I is explicitly declared to be type real. In the absence of this declaration, the variable type would be integer, and the real constant (1.345) would be truncated to an integer constant (1).

```
REAL   I   ;
I  =  1.345 ;
```

CHARACTER Command

Function: Declares a variable to be type character.

Syntax: CHARACTER variable, ... ;

Arguments: variable - name of a variable.

Description:

The CHARACTER command explicitly declares a variable to be type character. In an implicit declaration (see ASSIGN command), a character variable is declared by double quotes enclosing the value. Hence, the real purpose of the CHARACTER command is to allow character variables to be assigned other character variables. In this case, the double quotes are not entered.

Any number of character variables may be declared in a single command. If explicit declaration is used, it must precede the variable assignment and use. Once declared (implicitly or explicitly), the variable type can never be changed, but new values may be assigned to the variable, as convenient. The character variable can be used in place of a character constant wherever one is required as a data entry.

See Also: **ASSIGN, Ch. 6**
INTEGER, Ch. 6
REAL, Ch. 6

Examples:

This example illustrates some simple character variable operations.

```
CHARACTER alfa,beta;         ! alfa is explicitly character.
alfa = "This is a string.";! Quotes are still necessary.
beta  =  alfa;               ! Both are character variables, no quotes.
```

FUNCTION Command

Function: Creates functions for use by other commands.

Syntax:
FUNCTION function(x,...) = expression, ... ;
FUNCTION function DATA pairs x,y ... ;

Aguments:

function	- function name, user-defined.
x, ...	- first independent variable, etc., user-defined.
expression	- arithmetic expression in the FORTRAN style containing constants, variables, intrinsic functions, user-specified functions, the mathematical operators add (+), subtract (-), multiply (*), divide (/), and exponentiate (**), and the Boolean operators (.GT., .LT., .LE., .GE., .EQ., and .NE.).
pairs	- number of data pairs (integer).
x,y	- independent and dependent values of data pair (real).

Description:

The FUNCTION command is used to create a function and to give it a unique function name. The function can then be used in other commands simply by entering the name. Only the function name, and not the parentheses or independent variables, is entered. (The function name must be unique, and a previously-created function will be replaced by a new function if they have the same function name.)

The two syntactical forms allow use of either expressions or tabular data. An expression may contain up to ten independent variables. They must be enclosed by parentheses and separated by commas within the function name. Expressions are entered as strings in FORTRAN style. The equals sign shown in the command syntax must be entered.

As the syntax indicates, it is possible to associate more than one expression with a single function name. In this event, MCL creates a vector (multi-value) function. This capability can be used very effectively with the pseudo-array (multi-variable) capability described in the ASSIGN command (also see Examples, below).

Mathematical and Boolean operators have the same relative priority as in FORTRAN. The Boolean operators (.GT., .LT., .LE., .GE., .EQ., and .NE.) can be used to construct logical functions (see Examples, below). The result of a logical expression using Boolean operators is 1 if the expression is true and 0 if the expression is false. Two pre-defined system variables, SYS$TRUE (=1.0) and SYS$FALSE (=0.0), exist to facilitate Boolean operations.

The expression may contain various other existing functions, which include:

user-specified functions (previously created in other FUNCTION commands),
intrinsic mathematical functions (see Tables),
grid mesh and index functions (see Tables), and
feedback-loop functions (see Tables).

Any user-specified function (one previously created in a FUNCTION command) can be included in another FUNCTION command.

The Tables list all of the available in-line functions. The intrinsic mathematical functions include all of the standard trigonometric, transcendental, hyberbolic, numeric, and cylindrical Bessel functions. The grid mesh functions include coordinates as a function of indices and indices as functions of coordinates.

The Tables also list two feedback-loop functions, OBS$name and LASTKEY. The LASTKEY function is the ASCII code (integer value) of the last keystroke on the keyboard. This allows user feedback, i.e., the ability to control features of the simulation while it is in progress. Time-history data (see OBSERVE command) from the simulation itself may be entered as a function, thus providing a simulation feedback loop. Observer values are referenced as OBS$name, where "name" can be set in the OBSERVE command. The value of OBS$name when the function is computed is the value of the observer at the end of the previous time step. Examples using both types of feedback loops are given below.

For the DATA option, a set of (x,y) data pairs is entered in order of increasing x. This produces a function of a single independent variable, or y = function(x). When the function is evaluated for a value of x which falls between data values, y will be computed via linear interpolation. When it is evaluated for a value outside the data range, the nearest end-point value is used; i.e., extrapolation is never performed.

Restrictions:

1. A function must be defined before it is referenced by other commands.
2. A function may not reference itself.
3. A function whose name duplicates an intrinsic function will replace the intrinsic function.
4. Function names must not duplicate variable names.
5. The number of data pairs per function is limited to 300.

See Also: **ASSIGN, Ch. 6**
OBSERVE, Ch. 22

Examples:

1. The OBS$name function provides simulation feedback loop capability. We wish to use the dynamic voltage measured at Port_B to modify the incoming voltage applied at Port_A using a PORT command. The following commands could be employed.

```
OBSERVE FIELD E2 Port_B SUFFIX VOUT ;
FUNCTION VINC(T) = CONST - OBS$VOUT ;
PORT Port_A POSITIVE INCOMING VINC FUNCTION E1 g ;
```

2. The following commands create two functions. The first function, named ALFA, is a sine wave function of retarded time. The second function, named BETA, includes the first, modifying it with the addition of some noise.

```
C = 2.998E8 ;
FUNCTION ALFA(X,T) = 10. * SIN (4E9 * (X/C+T) ) ;
FUNCTION BETA(X,T) = ALFA(X,T) *  (1 + GAUSSIAN(0,.05));
```

3. A command syntax contains the symbol, f(t,x), indicating a requirement for two independent variables. However, the desired input is linear in t and independent of x. This can be entered as an expression or even by using the DATA option.

```
FUNCTION PULSE(T,X)= T / 1.0E-8 ;
FUNCTION PULSE DATA 2 0.0,0.0 1.0E-8,1.0 ;
```

Within the range (0<t<1e-8), these functions will give the same result. MCL ignores the unused independent variable from the first command. In the second command, MCL interprets the data as a function of the first independent variable (in this case, t) and performs linear interpolation between the data points. (The data option cannot be used with more than one independent variable.)

4. A vector (multi-value) function is created by entering multiple expressions in the same FUNCTION command. In the following example, a vector function having different linear rates is created, and the values evaluated at t = 1.0 are assigned to a (multi-variable) pseudo-array.

```
FUNCTION Pulse(t) = 1*t, 2*t, 3*t ;
X = Pulse(1.0);
```

These two commands create the variables and values equivalent to those produced in the following commands:

```
INTEGER X_DIM ;    X_DIM = 3 ;    X = 1.0 ;
X_1 = 1.0 ;         X_2 = 2.0 ;    X_3 = 3.0 ;
```

5. The LASTKEY function provides user feedback-loop capability. The following commands enable beam emission to be controlled interactively from the PC console by pressing the B-key (ASCII code 66). The function expression uses a Boolean logic operator, and the value is either 0 (false) or 1 (true). The beam is turned on when the B-key is pressed and turned off when any other key is pressed.

```
FUNCTION BEAM_ON(T) = LASTKEY.EQ.66 ;
FUNCTION CURRENT_J(T) = BEAM_ON(T) ;
EMISSION BEAM CURRENT_J VOLTAGE ;
```

Trigonometric Functions.

Function	Type	Argument	Definition
SIN(x)	real	real	Sine (x), argument in radians
COS(x)	real	real	Cosine (x), argument in radians
TAN(x)	real	real	Tangent (x), argument in radians
ASIN(x)	real	real	Arcsine (x), result in radians
ACOS(x)	real	real	Arccosine (x), result in radians
ATAN(x)	real	real	Arctan (x), result in radians
ATAN2(x,y)	real	Real, real	Arctan(x/y), result in radians

Transcendental Functions.

Function	Type	Argument	Definition
SQRT(x)	real	Real	Square Root: $x^{1/2}$
ELLIPTIC1(x)	real	Real	Complete elliptic integral of first kind: K(x)
ELLIPTIC2(x)	real	Real	Complete elliptic integral of second kind: E(x)
EXP(x)	real	Real	Exponential: e^x
IELLIPTIC1(ϕ,k)[1]	real	Real	Incomplete elliptic integral of first kind: $F(\phi\backslash\alpha)$ [1]
IELLIPTIC2(ϕ,k)[1]	real	Real	Inomplete elliptic integral of second kind: $E(\phi\backslash\alpha)$ [1]
IELLIPTIC3(ϕ,k,n)[1]	real	Real	Incomplete elliptic integral of third kind: $\Pi(n;\phi\backslash\alpha)$ [1]
LOG(x)	real	Real	Natural Logarithm: ln (x)
LOG10(x)	real	Real	Common Logarithm: log10 (x)

1. For the incomplete elliptic integrals, k=sin(α), such that $0^{\circ} \leq \alpha \leq 90^{\circ}$.

Numeric Functions.

Function	Type	Argument	Definition
ABS(x)	real	real	Absolute Value: $\|x\|$
INT(x)	integer	real	Truncated Integer: x = I + r, r < 1
MAX(x,y)	real	Real, real	Largest Value of x or y
MIN(x,y)	real	Real, real	Smallest Value of x or y
MOD(x,y)	real	Real, real	Remaindering: x-INT(x/y)*y
NINT(x)	integer	real	Nearest Int: INT(x+.5),x>0INT(x-.5),x<0
SIGN(x)	real	real	Sign of x; -1 (x<0), +1 (x>0)
GAUSSIAN(x,y)	real	Real, real	Gaussian Random No: $\exp(-(r-x)^2/2y)$
RANDOM	real	none	Uniform Random No; 0<r<1
STEP(x,y)	real	Real, real	Step Function: 1 (x>y), 1/2 (x=y), 0 (x<y)
THETA(x)	real	real	Step Function: 1 (x>0), 0 otherwise
RAMP(x)	real	real	Max(0,min(1,x))
SMOOTH_RAMP(x)	real	real	$\mathrm{Sin}(\pi/2*\mathrm{ramp}(x))^2$

Hyperbolic Functions.

Function	Type	Argument	Definition
SINH(x)	real	Real	Hyperbolic Sine: sinh (x)
COSH(x)	real	Real	Hyperbolic Cosine: cosh (x)
TANH(x)	real	Real	Hyperbolic Tangent: tanh (x)

Cylindrical Bessel Functions

Function	Type	Argument	Definition
BESSELJ0(x)	real	real	$J_0(a)$
BESSELJ1(x)	real	real	$J_1(a)$
BESSELY0(x)	real	real	$Y_0(a)$
BESSELY1(x)	real	real	$Y_1(a)$
BESSELI0(x)	real	real	$I_0(a)$
BESSELI1(x)	real	real	$I_1(a)$
BESSELK0(x)	real	real	$K_0(a)$
BESSELK1(x)	real	real	$K_1(a)$
BESSELJN(n,x)	real	Integer, real	$J_n(x)$
BESSELJP(n,x)	real	Integer, real	$J'_n(x)$
BESSELYN(n,x)	real	Integer, real	$Y_n(x)$
BESSELYP(n,x)	real	Integer, real	$Y'_n(x)$
BESSELJNZ(n,x)	real	Integer, real	Zeros of Bessel Function, $J_n(x)$
BESSELJPZ(n,x)	real	Integer, real	Zeros of Derivative, $J'_n(x)$
BESSELYNZ(n,x)	real	Integer, real	Zeros of Bessel Function, $Y_n(x)$
BESSELYPZ(n,x)	real	Integer, real	Zeros of Derivative, $Y'_n(x)$

Grid Mesh Functions

Function	Type	Argument	Definition
X1FG(i)	real	Integer	Nearest x_1 full-grid point vs. grid index
X2FG(i)	real	Integer	Nearest x_2 full-grid point vs. grid index
X3FG(i)	real	Integer	Nearest x_3 full-grid point vs. grid index
X1HG(i)	real	Integer	Nearest x_1 half-grid point vs. grid index
X2HG(i)	real	Integer	Nearest x_2 half-grid point vs. grid index
X3HG(i)	real	Integer	Nearest x_3 half-grid point vs. grid index
X1GR(x)	real	Real	X_1 coordinate vs. real (continuous) grid index
X2GR(x)	real	Real	X_2 coordinate vs. real (continuous) grid index
X3GR(x)	real	Real	X_3 coordinate vs. real (continuous) grid index

Grid Index Functions

Function	Type	Argument	Definition
I1FG(x)	integer	real	Grid index vs. nearest x_1 full-grid point
I2FG(x)	integer	real	Grid index vs. nearest x_2 full-grid point
I3FG(x)	integer	real	Grid index vs. nearest x_3 full-grid point
I1HG(x)	integer	real	Grid index vs. nearest x_1 half-grid point
I2HG(x)	integer	real	Grid index vs. nearest x_2 half-grid point
I3HG(x)	integer	real	Grid index vs. nearest x_3 half-grid point
I1CL(x)	integer	real	Grid index of cell containing x in x_1
I2CL(x)	integer	real	Grid index of cell containing x in x_2
I3CL(x)	integer	real	Grid index of cell containing x in x_3

Feedback Loop Functions

Function	Type	Argument	Definition
LASTKEY	integer	none	ASCII code of last key stroke by user
OBS$name	real	none	Time-history data from simulation

This page is intentionally left blank.

7. CONTROL STATEMENTS

This Chapter covers the following commands:

DO / ENDDO
IF / ELSEIF / ELSE / ENDIF
CALL / RETURN
$namelist$

You can use these commands to control the flow of commands to be processed.

The DO / ENDDO commands provide the capability to loop repeatedly over the same block of commands. They function in a manner similar to their FORTRAN counterparts. The counter variable and the increment can be real as well as integer, and the counter increment can be negative as well as positive. Unlike FORTRAN, the counter can be reset within the loop to provide an exit.

The logical IF commands allow decision making and branching within the input file. These commands also follow the FORTRAN convention. Any number of ELSEIF commands can be used within the IF / ENDIF pair; however, only one ELSE command is allowed.

The CALL / RETURN construct provides a pseudo-subroutine capability that allows a single, large input file to be broken up into smaller, more convenient files. This allows better organization of the input data by function. It also avoids unnecessary duplication of input data, since any data which repeats can simply be placed in a separate file and called repeatedly. The input data contained in one file is available to all the other files (there is no hidden input data).

Finally, the $namelist$ command provides a facility to process existing namelist files without altering them. This allows input data produced for other applications to be read directly.

DO / ENDDO Commands

Function: Repeats execution of a block of commands.

Syntax:

```
DO  variable = expression, expression [ , expression ] ;
        [ block ]
ENDDO ;
```

Arguments:

variable	-	do-loop counter variable of type integer or real.
expression	-	scalar expression of type integer or real. The first expression is the initial value of the variable, the second is the final value, and the third (optional) is the increment (default = 1).
block	-	a sequence of executable commands.

Description:

The DO / ENDDO construct allows repetition of a blocked set of commands. The block must be placed between the DO and ENDDO commands. The variable is the do-loop counter, which may be real or integer. Its initial, final, and increment values are evaluated from the three expressions. If the third expression is not entered, then the increment is given a default value of unity. The block is repeated until the variable passes the final value (the increment can be negative with the variable decreasing).

DO / ENDDO constructs can be nested. One do-loop can lie within another, as long as the range of the inner loop lies completely within the range of the outer DO loop. Each of the nested loops should have a unique variable.

The value of the do-loop counter variable can be changed within the loop, which provides a useful means for exiting the loop. This is typically associated with a logical IF test on some variable calculated within the loop. To cause an exit, simply set the variable to a value beyond the final value.

Restrictions:

Do-loop nesting is limited to a depth of eight.

Examples:

1. Do-loops can be used for many different purposes. This example demonstrates the creation and unique labeling of ten repeating area "islands" which might be used to construct ridges in a waveguide. Given the island width w, spacing d, and height h, we have

```
Xa  =  - d  ;
Ya  =  0  ;
Yb  =  Ya  +  h  ;
Nislands  =  10  ;
DO I  =  1,  Nislands  ;
      Xa  =  Xa  +  d  ;
      Xb  =  Xa  +  w  ;
      AREA  Island'I'  RECTANGULAR  Xa,Ya  Xb,Yb  ;
ENDDO  ;
```

The resulting areas are named "Island1, Island2, ... Island10."

2. This example inverts a transcendental equation using Newton's method. It uses do-loop variable reassignment to terminate the loop when convergence is achieved.

```
! Problem: solve f(x) = x exp(x) = 10 for x.
FUNCTION f(x) = x * exp (x) ;
y = 10. ;  x_guess = 1.0 ;  dx = 0.001 * x_guess ;
tolerance = 0.0001 * x_guess ;          ! four decimal points accuracy.
DO iter = 1, 1000 ;
     x_solution = x_guess -
          ( f(x_guess) - y ) * dx / (f(x_guess + dx) - f(x_guess) ) ;
     IF ( ABS (x_solution - x_guess) .LT. tolerance ) THEN ;
          iter = 1001 ;
     ENDIF ;
     x_guess = x_solution ;
ENDDO ;
! Returns with x_solution = 1.7455.
```

IF / ELSEIF / ELSE /ENDIF Commands

Function: Provides conditional execution of commands.

Syntax:

```
IF ( expression ) THEN ;
        [ block ]
[ ELSEIF ( expression ) THEN ;
        block ]
        ...
[ ELSE ;
        block ]
ENDIF ;
```

Arguments:

expression - logical expression.
block - sequence of executable commands.

Description:

The IF command works together with ELSE, ELSEIF, and ENDIF commands to provide conditional execution of other commands. As the syntax indicates, the ELSE and ELSEIF branches are optional. Multiple ELSEIF branches can be entered between the IF and ENDIF commands, but no more than one ELSE branch can be used.

In writing a logical expression, mathematical and Boolean operators have the same relative priority as in FORTRAN. The Boolean operators, .GT., .LT., .LE., .GE., .EQ., and .NE. can be used to construct logical expressions. The result of a logical expression is 1 if the expression is true and 0 if the expression is false. Two pre-defined system variables, SYS$TRUE (=1.0) and SYS$FALSE (=0.0), exist to facilitate Boolean operations.

Examples:

In the following example, the number of time steps, Kmax, is calculated from variables Kcycle and Ncycles, which have been entered previously in the input data. (The variable Kcycle is the number of time steps in an RF cycle, and Ncycles is the number of RF cycles.)

```
IF ( Ncycles .EQ. 0 ) THEN ;
      Kmax = 1 ;
ELSE IF ( Ncycles .GT. 0 ) THEN ;
      Kmax = Ncycles * Kcycle ;
ENDIF ;
```

Notice that a negative value of Ncycles will result in Kmax being undefined unless it is assigned elsewhere.

CALL / RETURN Commands

Function: Opens, processes, and exits a command file.

Syntax: CALL new _file ; (in the first file)

RETURN ; (at the end of new_file)

Arguments: new_file - name of a second command file.

Description:

The CALL / RETURN commands provide pseudo-subroutine capability. It allows a single input file to be broken up into a number of smaller files, which can be organized according to function. For example, it may be desirable to separate design parameter data from simulation commands. Also, commands which would be repeated many times in an input file are clearly candidates for this treatment.

A CALL command in one file transfers control to "new_file". (All of the constants, variables etc., from either file are accessed by the other, so there is no list of arguments to be transferred between files.) The new_file can contain any commands which could be used in the original file. It can also contain logical IF commands which lead to one or more RETURN commands. The last command in new_file should be a RETURN command. If no RETURN is encountered in new_file, an automatic return will be generated when the end of the file is encountered. When any RETURN command is encountered, new_file will be closed, and commands after the CALL in the original file will be processed.

CALL commands may be nested up to six levels deep. Thus, new_file may also contain CALL commands to other files, etc.

Restrictions:

1. CALL commands cannot be nested more than six levels deep.
2. Any file which is accessed with a CALL command should be terminated with a RETURN command.
3. CALL commands are not allowed within do-loops.

See Also: **$namelist$, Ch. 7**

Examples:

In developing a template for simulation of a generic device, one might wish to isolate the actual input data for a specific design from the generic simulation (algorithm, output, etc.) commands. Thus, at the point in the template where design data is required, the command

```
CALL design.mgc ;
```

could be inserted. This file, design.mgc, would then contain the design-specific data (and be terminated with a RETURN command). The advantage of this approach is that the same simulation methodology can be applied to many different designs which are isolated and identified in individual files.

$namelist$ Command

Function: Processes FORTRAN-style namelists.

Syntax:

```
$group_name
        [ block ]
$[end]
```

Arguments:

group_name	- symbolic name of namelist.
block	- block of variable = value assignments.
end	- terminator on some namelists (ignored by MCL).

Description:

The $namelist$ command is provided to facilitate data entry from FORTRAN-style namelists. (Use of this command for any other purpose is discouraged.) It allows a namelist file to be read simply by using the CALL command to access the file. It should not be necessary to alter the namelist file itself.

The namelist file may contain more than one namelist. In each namelist, the block between $group_name and $end should consist of the usual FORTRAN-style assignments, i.e., variable = value. Assignments should be separated by commas. Array and character assignments are permitted.

MCL initially treats each namelist as a single MCL command. The command begins and ends with the namelist delimiter ($). (The DELIMITER command can be used to change the default delimiter ($) to accommodate an unusual namelist convention.) Everything to the right of the second delimiter is ignored by MCL. Hence the "end" in $end is ignored. (Thus, it is not necessary to add a RETURN command to a namelist file, even though it is accessed by a CALL command.)

MCL accommodates multiple occurrences of the same namelist in a file by using a pseudo-array capability for group_name variables. It will also create a variable named group_name_INSTANCES to record the number of such occurrences. It will treat any namelist arrays which it encounters in a similar manner. (The user is encouraged to become familiar with the variable-naming convention in order to be able to use the namelist data effectively in other MCL commands. See Examples, below).

See Also: CALL, Ch. 7
DELIMITER, Ch. 8

Examples:

A popular 1D helix-TWT program requires input from a namelist file. This file, named HELIX.NML, includes the following lines:

```
$helix   name="input helix", radius=1.,
pitch=.300,.315
$end
$helix   name="output helix",
radius=1.,
pitch=.315,.295,.285
$end
```

The MCL input file contains a single command to read this namelist file:

```
CALL HELIX.NML ;
```

The processing of this namelist file will produce variables and values equivalent to those that would be obtained by executing the following commands:

```
INTEGER HELIX_INSTANCES ;
HELIX_INSTANCES = 2 ;
HELIX_1.NAME = "input helix" ;
HELIX_1.RADIUS = 1. ;
INTEGER HELIX_1.PITCH_DIM ;
HELIX_1.PITCH_DIM = 2 ;
HELIX_1.PITCH = .300 ;
HELIX_1.PITCH_1 = .300 ;
HELIX_1.PITCH_2 = .315 ;
HELIX_2.NAME = "output helix" ;
HELIX_2.RADIUS = 1. ;
INTEGER HELIX_2.PITCH_DIM ;
HELIX_2.PITCH_DIM = 3 ;
HELIX_2.PITCH = .315 ;
HELIX_2.PITCH_1 = .315 ;
HELIX_2.PITCH_2 = .295 ;
HELIX_2.PITCH_3 = .285 ;
```

This page is intentionally left blank.

8. I/O UTILITIES

This Chapter covers the following commands:

BLOCK / ENDBLOCK
COMMENT / C / Z / !
DELIMITER
ECHO / NOECHO

You can use these commands to modify the input file and to control disposition of the processed commands.

The BLOCK / ENDBLOCK commands allow the transfer of blocks of text from the input file to any other file. It can provide substantial automation for simulations which involve more than one code. For example, the input file for a simulation can automatically create the input file required for post-processing. It can even submit the post-processing run automatically. The BLOCK / ENDBLOCK commands can also be used to wrapper codes.

MCL supports extensive commenting capability with the COMMENT commands. Combined with the use of (self-documenting) variables, input files can easily be fully documented. In addition, you can use this capability to disable commands temporarily without actually removing them from the input file.

You can use the DELIMITER command to redefine any of the delimiter characters. For example, if an existing namelist file contains a non-standard delimiter, you can simply re-define the MCL delimiter, whereas changing the file itself might render it unreadable in its original application. Another convenient use of this command is to redefine the command delimiter (default ;) as a carriage return. This would potentially allow actual FORTRAN to be processed.

The ECHO / NOECHO commands provide control over what goes into the output (LOG) file to prevent the huge output files which would result from processing commands within a do-loop, for example. The JOURNAL command can be used to record commands entered interactively. Thus, a file can be developed interactively and saved for subsequent runs in the batch mode.

BLOCK / ENDBLOCK Commands

Function: Copies blocks of text from the input file to some other file.

Syntax:

```
BLOCK { COMMENT,
        WRITE  file [ write_mode [ carriage_control ] ] }
    [ block ]
ENDBLOCK ;
```

Arguments:

file	- name of the other file.
write_mode	- file write mode (NEW(default), OVERWRITE, or APPEND).
carriage_control	- carriage control for file (FORTRAN (default) or LIST).
block	- arbitrary text.

Description:

The primary use of the BLOCK / ENDBLOCK command is to copy a block of text from the input file to some other file. It provides a convenient means of creating other input files for post-processing, etc. Text is copied verbatim, with one exception; any designated variable substitution is performed before the text is copied (see Examples, below).

Following the BLOCK keyword, you must select either COMMENT or WRITE. The COMMENT keyword copies only to the output (LOG) file, while the WRITE keyword copies to any user-specified file name. In the latter case, write_mode specifies what will happen if the file already exists. If you enter NEW (the default), the system tries to open a new file. It will not overwrite or destroy an existing file; instead, the BLOCK command returns with an error message. Entering OVERWRITE will destroy an existing file. Entering APPEND adds to the existing file and is useful for constructing files a bit at a time.

The carriage_control provides carriage control in the new file. If you enter FORTRAN, a blank character will be placed at the beginning of each line (required for certain FORTRAN programs), whereas LIST will cause each line to start in column one. Any variable substitution (designated by appropriate delimiters, see ASSIGN command) will be performed before the text is copied. The block of text itself is delimited by the ENDBLOCK command.

The BLOCK commands can provide substantial automation. For example, an input file for one code, e.g., MAGIC, can also write an input file for post-processing in another code, e.g., POSTER. It can even submit the post-processing job. Variable substitution allows important data, e.g., a beam voltage or an antenna current, to be passed from the MAGIC run to the subsequent post-processing run. Ultimately, the combination of the BLOCK command, variable substitution, and the COMMAND command give MCL very powerful wrappering capability, not just for the MAGIC family of codes, but for any set of codes.

See Also: ASSIGN, Ch. 6
COMMAND, Ch. 9

Examples:

The following example writes an input file for one of the POISSON magnet design codes. It also writes a DOS batch file to run the POISSON code. Finally, it submits a DOS command to run the batch file. (The variables, TITLE, IRUN, DX, RMAX, and ZMAX have been previously defined.)

```
! === write AUTOMESH input file ===
BLOCK WRITE "''IRUN'.DAT" OVERWRITE ;
'TITLE'
$REG NREG=1, DX='DX', XMAX='RMAX', YMAX='ZMAX', NPOINT=5,  ITRI=2 $
$PO X=0.0 , Y=0.0  $
$PO X=0.0 , Y='ZMAX'  $
$PO X='RMAX' , Y='ZMAX'  $
$PO X='RMAX' , Y=0.0  $
$PO X=0.0 , Y=0.0  $
ENDBLOCK ;
! === DOS batch file to run AUTOMESH ===
BLOCK WRITE "''IRUN''.BAT" OVERWRITE ;
AUTOMESH < 'IRUN'.DAT  >> AUTOMESH.LOG
COPY  OUTAUT  'IRUN'.AUT
COPY  TAPE73  'IRUN'.T73
DEL OUTAUT
DEL TAPE73
ENDBLOCK ;
! === Run batch file ===
COMMAND "'IRUN'.BAT" ;
```

COMMENT / C / Z / ! Commands

Function: Creates comments in input and/or output files.

Syntax: COMMENT "comment" ;

C [command] ;

Z [command] ;

[command] ! [comment]

Arguments:

comment - user-defined character string.
command - command name plus data entries.

Description:

MCL allows comments to be entered in the input file in a variety of ways. Some, but not all, are echoed in the LOG file.

The COMMENT command is intended to enter general comments in the input file which are echoed in the LOG file. The presence of a COMMENT command cannot affect the behavior of the simulation. The comment must be enclosed by double quotes. Thus, any character except double quotes is permitted within the comment. For example, a semicolon is permitted within the comment. Text is copied verbatim, with one exception; any designated variable substitution is performed before the text is copied. (Also, see BLOCK COMMENT command, which does not require double quotes.)

The C command is typically used to preserve another command in the input file without executing it. Because the original command is now ignored, the simulation results may change. The C command can extend over many lines and is terminated by the command delimiter. Simply adding the character C to the beginning of an existing command typically creates it.

The Z command is also used to preserve a command in the input file without executing it. Thus, it too can affect the behavior of the simulation. It can extend over many lines and is terminated by the command delimiter. (The only difference between the C and Z commands is that the Z writes the message "Command Ignored" in the LOG file, whereas C writes nothing.)

The ! command is typically used to enter short comments following another command on a single line of the input file. The comment is not echoed in the output file. Thus, ! is similar to C. They differ in that C may extend over many lines (before the command delimiter), while ! applies only to the remainder of a single line (and has no delimiter). Note that an executable command may extend over many lines, and any or all of the lines may contain a ! comment.

See Also: **ASSIGN, Ch. 6**
BLOCK, Ch. 8

Examples:

The commands below depict various uses of the comment commands.

```
COMMENT  "The C command below preserves the ASSIGN command in the
input file, but no information will be written to the output file.
(By contrast, Z would write, Command Ignored.)  In either case, the
command will be ignored (not processed). Note that, as written, the
command below extends over two lines."  ;

C ALFA
     = 100;

! This comment extends only to the end of the line.
! Every line needs a new ! command. No delimiter or echo in output.
! You can comment out another command if it is all on one line.
! An example is:  TITLE "This is a title" ;
! (This TITLE command will not be processed.)
! However, the ASSIGN command below will be processed.

ALFA  =  100.  !  This comment might explain why ALFA is set to 100.

! Note that a command with ! comments may be spread over many lines,
! e.g.,

FUNCTION  Big_wave(x,t)  =        !  This line has the function name.
     SIN (k*x - w*t)  ;           !  This line has the expression.

!  The following COMMENT command illustrates variable substitution.

A  =  5. ;
COMMENT  " The value of A is 'A'   "  ;

!  The comment will read " The value of A is 5.00e+00 "
```

DELIMITER Command

Function: Redefines delimiters.

Syntax: DELIMITER [type] character ;

Arguments:

type - type of delimiter (list below).
COMMAND (default), default character is semicolon (;)
SUBSTITUTION, default character is single quote (')
STRING, default character is double quote (")
NAMELIST, default character is dollar sign ($)
FORMAT, default character is colon (:)

character - ASCII character (RETURN, or select from character set).

Description:

The DELIMITER command is used to change MCL delimiters. The new delimiter will take effect on the following command. Delimiters may be changed as often as desired.

The type specifies which delimiter to change. A delimiter can be changed to virtually any other single character. If more than one character is entered, then the first character is used, and the remaining characters are ignored. The single exception to this rule occurs for the COMMAND delimiter, which may be set to a carriage return as described below.

The default value for the COMMAND delimiter is a semicolon (;). You can change this to a carriage return by entering the character constant, RETURN. With this delimiter, command continuation is provided with an (&) at the end of each line. (Alternatively, a long mathematical expression can be entered without continuation characters by enclosing the expression in double quotes.) The semicolon delimiter for commands remains active in addition to any new delimiter specified.

Restrictions:

Delimiters should not be set to a comma or to a period.

Examples:

1. The COMMAND delimiter can be set to a slash (/) to allow use of older MCL input decks, written when the default delimiter was a slash.

```
DELIMITER   COMMAND  /  ;
C  Then the following old commands don't need to be changed. /
TITLE "TEST CASE 1" /
      ...
C  Reset the delimiter again  /
DELIMITER  RETURN  ;
C  This allows entering a command without using a delimiter; however,
& must precede continuation lines
```

2. The following commands illustrate how to create and use character variables to redefine the symbol substitution delimiter and the character string delimiter. First, character variables are assigned, and then the

default SUBSTITUTION delimiter is replaced with the character %, and the default STRING delimiter is replaced with the character &. Next, the character variable SINGLEQUOTE is defined to be the character of the same name, as is the character variable DOUBLEQUOTE. Finally, the delimiters are returned to their original default values.

```
CHARACTER   SINGLEQUOTE  ;
CHARACTER   DOUBLEQUOTE  ;
DELIMITER   SUBSTITUTION   %  ;
DELIMITER   STRING   &  ;
SINGLEQUOTE   =   &'&  ;
DOUBLEQUOTE   =  &"&  ;
DELIMITER   STRING   %DOUBLEQUOTE%   ;
DELIMITER   SUBSTITUTION   %SINGLEQUOTE%    ;
```

3. The FORMAT delimiter is used for variable substitution with a specific format. The example below uses both the SUBSTITUTION and FORMAT delimiters.

```
BEAM_VOLT = 130.354Kilovolts ;
VOLTS = BEAM_VOLT/1.kilovolt ;
ASTRING = "Beam voltage is 'VOLTS:F4.0' kilovolts." ;
```

The character variable ASTRING will have the value "Beam voltage is 130 kilovolts." Another example is given below in which a file name is created which includes a run number padded with zeros.

```
IRUN = 2 ;
AFILE = "RUN'IRUN:I2.2'" ;
```

In this case, the character variable AFILE will have the value "RUN02".

ECHO / NOECHO Commands

Function: Turns on / off the echoing of processed commands to the LOG file.

Syntax:

NOECHO ;
    [commands]
ECHO ;

Arguments: commands - arbitrary commands.

Description:

The ECHO command enables echoing output of processed commands to the LOG file. This is the default condition. The NOECHO command disables echoing of the processed commands to the LOG file.

Examples:

This input demonstrates the use of ECHO and NOECHO. Within the do-loop, NOECHO and ECHO are used to suppress output from certain commands as unnecessary. For the commands between the NOECHO and the ECHO, output of the processed commands will only occur if there is an error in a command.

```
DO I = 1, 100 ;
    NOECHO ;                             ! Turn off output.
        [first block of commands]        ! Output is suppressed.
    ECHO ;                               ! Turn output back on.
        [second block of commands]       ! Output is on.
ENDDO ;
```

9. EXECUTION CONTROL

This Chapter covers the following commands:

START / STOP
TERMINATE
COMMAND
KEYBOARD

You can use these commands to control execution of the input file.

The most commonly used commands are START, which initiates the simulation after all input data has been entered, and STOP, which terminates the application.

You can use the TERMINATE command to specify circumstances under which you want the simulation to stop. The options range from none to errors of decreasing severity. Novice users often benefit by resetting the termination to ERROR level, instead of to the default ABORT level.

The COMMAND command allows you to send a command to the operating system while your MCL application is still running. It can be used to start other applications from your MCL application. The KEYBOARD commands also allow you to interact with the simulation during execution.

START / STOP Commands

Function: Interrupts the processing of input and initiates / terminates the simulation.

Syntax:

```
START ;
STOP ;
```

Arguments: None.

Description:

The START command initiates execution of the simulation immediately after processing this command. If there are errors in any of the prior commands, the simulation will be terminated.

The STOP command terminates execution immediately after processing this command. Any commands that follow the STOP command will be ignored.

TERMINATE Command

Function: Terminates execution on errors of specified severity.

Syntax: TERMINATE severity ;

Argument: severity - error severity level (list below).
NONE - do not terminate.
ABORT - terminate on abort (default).
ERROR - terminate on abort or error.
WARNING - terminate on abort, error, or warning.

Description:

This command allows the user to control termination on errors of different levels of severity. Specifying severity causes termination for all errors of that and all greater severity. Termination occurs as soon as the error condition is encountered. No further input processing takes place.

The error conditions are listed above in order of increasing severity, as follows. The NONE level will attempt to ignore all errors and continue the simulation. ABORT conditions are generally due to code errors. ERROR conditions are usually caused by user input errors that result in failure. WARNING conditions are caused by user input that is legal, but in violation of the intended usage.

Novice users can benefit the most by setting the termination level to TERMINATE ERROR. This usually causes a termination very near to where a problem is. More advanced users who wish to detect more than one error per run will probably wish to use the default ABORT level termination.

Restrictions:

Not all errors are categorized by severity level. These errors will not be captured by this command.

See Also: **MESSAGE, Ch. 9**
PAUSE, Ch. 9

Examples:

If a variable is not defined prior to being used, an error is generated. In the following example, the variable, VOLTMAX, has been misspelled as VLTMAX. Thus, this variable is undefined in the function, VOLTIME, and an error occurs when the function is used in the next line. The TERMINATE ERROR setting causes MAGIC to stop execution at this line. If TERMINATE ABORT were used instead, then MAGIC would log the error, but continue processing.

```
TERMINATE ERROR ;
VOLTMAX = 50E3 ;
WOMEGA = 10.7E10 ;
FUNCTION VOLTIME(T) = VLTMAX * SIN ( WOMEGA*T ) ;
V10 = VOLTIME(10_nanoseconds) ;
```

COMMAND Command

Function: Issues a command to the operating system.

Syntax: COMMAND "system command"

Arguments: system command - operating system command in double quotes.

Description:

The COMMAND command sends a command to the operating system for immediate processing while the MCL application is still running. This allows other applications to be started from MAGIC.

Examples:

A template contains some design variables which the user needs to set before they are processed. These variables are in a file named DESIGN.MGC. The following commands bring up the file in the DOS editor, forcing the user to edit and save the file before it is processed by the CALL command.

```
COMMAND  "edit design.mgc" ; !  Forces user to edit file.
CALL  design.mgc  ;          !  Processes the file
```

KEYBOARD Commands

Function: Allows control key interaction with simulation during execution.

Syntax: N. A.

Arguments: N. A.

Description:

You may interact with a simulation during execution by pressing certain designated control keys. This allows examination of intermediate results while the simulation is still in progress.

There are two graphical output modes: window and file. The system is always in one mode or the other, and the mode selected determines whether graphical output appears on the monitor (window mode) or whether it is set to the PS graphics metafile (file mode). (See the GRAPHICS Command in Chapter 20). The designated control keys and their functions are as follows:

Esc — terminates simulation being executed
f3 — turns PAUSE ON for graphics (screen mode only)
f4 — turns PAUSE OFF for graphics (screen mode only)
f5 — displays all observe plots up to current time step
f7 — saves a bit map of the currently displayed plot in a subfolder named BITMAP
f8 — displays all phase-space plots
f9 — displays all contour plots
f10 — displays all perspective plots
f11 — displays all range plots
f12 — displays all vector plots

In addition to the designated control keys, you can use an intrinsic function named LASTKEY in commands to provide customized interactive capability (FUNCTION, Ch. 6).

Restrictions:

1. KEYBOARD commands are available only for the PC.

2. Plots that require an accumulation of data over several time steps will not be displayed unless the interrupt time matches that of the timer.

See Also: **FUNCTION (LASTKEY), Ch. 6**
TIMER, Ch. 11
GRAPHICS FILE, Ch. 20
GRAPHICS WINDOW, Ch. 20

This page is intentionally left blank.

10. OBJECTS

This Section covers the following commands:

SYSTEM
POINT
LINE
AREA
VOLUME
DELETE
LIST

You can use these commands to choose a coordinate system, to create spatial objects, to delete objects created previously, and to list objects for inspection.

The choice of coordinate system determines not only the system, but also the identification with generalized coordinates, as shown in the following table. This identification is important, because the command syntax uses the generalized coordinate notation.

System	(x_1, x_2, x_3)
cartesian	(x, y, z)
cylindrical	(z, r, ϕ)
polar	(r, ϕ, z)
spherical (2D)	(r, θ, ϕ)
spherical (3D)	(θ, ϕ, r)

In 2D simulations, the third coordinate (x_3) is ignorable. This has several important ramifications. First, the four systems are unique (in 2D); hence, all four are available in 2D simulations. Also, the third (ignorable) coordinate can, for the most part, be ignored. For example, it is not necessary to create a spatial grid in the third coordinate (GRID, Ch. 11). Two (not three) values are sufficient to define a point in x_1, x_2 space (POINT, Ch. 10). While we also have lines (LINE, Ch. 10), the most complicated spatial object in x_1, x_2 space is the area (AREA, Ch. 10). Spatial objects which have depth are not required; indeed, volume objects are not accepted in 2D simulations.

In 3D simulations, no coordinate is ignorable. Thus, spatial grids are required for all three coordinates, and it take three values (not two) to define a point. In addition to points, lines, and areas (all of which can exist in 2D simulations), we also have objects with depth called volumes (VOLUMES, Ch. 10). By inspection of the table above, note that the cylindrical and polar systems are identical (in 3D) except for the order of the coordinates. Therefore, the cylindrical and polar systems are redundant in 3D, and the choice is a matter of convenience.

As described above, there are four general types of spatial objects: points, lines, areas, and volumes. In addition, each type may include different shape options. For example, lines may be either straight or conformal (with a coordinate). Areas may be rectangular, conformal, or functional. (The conformal option is especially useful to create curved lines and areas in non-cartesian coordinate systems.) Similarly, volumes may be created in a variety of standard geometrical shapes, e.g., spheres, cylinders, etc.

Whenever you create a spatial object, you must give it a name. This allows the object to be referenced in other commands. For example, you can use a VOLUME command to create and name an object, and then use

the name to specify the object permittivity (DIELECTRIC, Ch. 14). Generally, the name should be a unique alphanumeric label which has meaning to you. If two or more objects are given the same name, an error message will result.

You can DELETE spatial objects which have already been created, and you can LIST the existing objects.

On loading, MAGIC automatically generates a system variable that indicates whether you are using the 2d or the 3d code, specifically, ISYS$CODE=2 for MAGIC2D and ISYS$CODE=3 for MAGIC3D. You may use these variables with conditionals to provide different processing paths for your input file.

MAGIC2D creates a system-generated area named OSYS$AREA that spans the mesh. This area object is created when the grid is generated. The area represented is the simulation area in the active coordinate system. It is updated each time the grid is modified. MAGIC3D creates three system-generated areas named OSYS$MIDPLANE1, OSYS$MIDPLANE2, and OSYS$MIDPLANE3. Each area spans the midplane of the x1, x2, and x3 coordinates respectively. In addition, MAGIC3D creates a system-generated volume named OSYS$VOLUME, which spans the defined grid. In all cases, the system-generated areas and volumes are of the type CONFORMAL. These defined areas and volumes may be used exactly, as a user defined area or volume. Just note that these are generated after the grid has been defined and are updated to maintain a match to the edges of the mesh definition.

SYSTEM Command

Function: Specifies the coordinate system.

Syntax: SYSTEM system ;

Arguments: system - coordinate system (list below).

CARTESIAN	- (x,y) in 2D,	(x,y,z) in 3D.
CYLINDRICAL	- (z,r) in 2D,	(z,r,ϕ) in 3D.
POLAR	- (r,ϕ) in 2D,	(r,ϕ,z) in 3D.
SPHERICAL	- (r,θ) in 2D,	(θ,ϕ,r) in 3D.

Defaults:

The default coordinate system for both 2D simulations and 3D simulations is CARTESIAN. The choice of system also determines identification with the generalized coordinates (x_1, x_2, x_3).

Description:

The choice of coordinate system determines not only the system, but also the identification with generalized coordinates, as shown in the following table. This identification is important, because the command syntax uses the generalized coordinate notation.

System	(x_1, x_2, x_3)
cartesian	(x, y, z)
cylindrical	(z, r, ϕ)
polar	(r, ϕ, z)
spherical (2D)	(r, θ, ϕ)
spherical (3D)	(θ, ϕ, r)

In 2D simulations, the third coordinate (x_3) is ignorable. This has several important ramifications. First, the four systems are unique (in 2D); hence, all four are available in 2D simulations. The third (ignorable) coordinate can, for the most part, be ignored. For example, it is not necessary to create a spatial grid in the third coordinate (GRID, Ch. 11). Two (not three) values are sufficient to define a point in x_1, x_2 space (POINT, Ch. 10). While we also have lines (LINE, Ch. 10), the most complicated spatial object in x_1, x_2 space is the area (AREA, Ch. 10). Spatial objects which have depth are not required; indeed, volume objects are not accepted in 2D simulations.

In 3D simulations, no coordinate is ignorable. Thus, spatial grids are required for all three coordinates, and it takes three values (not two) to define a point. In addition to points, lines, and areas (all of which can exist in 2D simulations), we also have objects with depth called volumes (VOLUMES, Ch. 10). By inspection of the table above, note that the cylindrical and polar systems are identical (in 3D) except for the order of the coordinates. Therefore, the cylindrical and polar systems are redundant in 3D, and the choice is a matter of convenience.

Restrictions:

1. The spherical coordinate system cannot extend to zero radius in either 2D or 3D simulations. However, the polar angle coordinate (θ) can extend to zero (or π) if azimuthal symmetry is imposed (SYMMETRY, Ch. 12).

2. The polar (r,ϕ, z) coordinate system cannot extend to zero radius in 2D simulations.

3. Axial symmetry boundary conditions are not automatically provided by the choice of coordinate system, but must be specified explicitly by the user.

See Also:
POINT, Ch. 10
LINE, Ch. 10
AREA, Ch. 10
VOLUME, Ch. 10
AUTOGRID, Ch. 11
GRID, Ch. 11
SYMMETRY, Ch. 12

References:

B. Goplen, R. Worl, and R. E. Clark, "Simulations of the PBFA-II Voltage Adder," Mission Research Corporation Report, MRC/WDC-R-091, November 1984.

Examples:

1. In this 3D example, the (default) cartesian system is automatically active. We will simply create some points in this cartesian coordinate system.

```
POINT Point_a Xa,Ya,Za ;
POINT Point_b Xb,Yb,Zb ;
```

2. In this 2D example, the coordinate system is polar, and we will create one point in this system.

```
SYSTEM  POLAR ;
POINT one_point Radius,Theta ;
```

POINT Command

Function: Specifies a point in space.

Syntax: POINT point x1, x2 [, x3] ;

Arguments:

point	- name of spatial object, user-defined.
x1, x2, x3	- spatial coordinates (m or rad).

Description:

The POINT command is used to specify the location of a point in space. The point name provides a unique label for the point. In other commands which require a point, you can enter either raw coordinate values or the name of a point previously defined in a POINT command. If a point name is duplicated in another POINT command, the original point data will be lost. If desired, variable substitution may be used to "subscript" similar spatial points (see Section 4.4).

In general, 2D simulations require two real coordinates (x1, x2) to define a point in space completely, while 3D simulations require three real coordinates (x1, x2, x3). This is indicated in the syntax by the option, [,x3], which means that a third coordinate is required for 3D simulations.

Restrictions:

1. No two points can have the same point name; if a point name is duplicated, the original spatial data will be lost, and an error message will be produced.

2. A 2D simulation requires only two coordinates to be entered for each point.

3. A 3D simulation requires three coordinates to be entered for each point.

See Also:

SYSTEM, Ch. 10
LINE, Ch. 10
AREA, Ch. 10
VOLUME, Ch. 10
DELETE, Ch. 10
LIST, Ch. 10

Examples:

In the examples which follow, it is assumed that the cartesian coordinate system is active unless otherwise noted.

1. To enter a spatial point named "origin" at the origin of a 2D cartesian coordinate system, the command is

```
POINT origin 0,0 ;
```

In a 3D simulation, the command would be

```
POINT origin 0,0,0 ;
```

Note that this does not define the coordinate system or the origin of a coordinate system. It is just a point named "origin."

2. To enter the two end-points named "a" and "b" defining the axis of symmetry (running from 0 to Zmax in z) in a polar (r,ϕ,z) coordinate system, some possible commands are

```
SYSTEM POLAR ;
POINT a 0,0,0 ;
POINT b 0,0,Zmax ;
```

Again, these are just points with names. To create symmetry, we require another command (SYMMETRY, Ch. 12).

3. To enter a series of 100 points starting at z_start and spaced equally every dz along the z-axis, consider using variable substitution within a do-loop, as illustrated in the following commands.

```
z = z_start - dz ;
DO i = 1, 100 ;
     z = z + dz ;
     POINT a'i' x,y,z ;
ENDDO ;
POINT b 0,0,Zmax ;
```

The result will be a series of points labeled a1, a2, a3, etc., each at a different axial location.

LINE Command

Function: Specifies a line in space.

Syntax:

```
LINE  line
      { CONFORMAL  point1  point2... ,
        OBLIQUE  point1  point2... ,
        CIRCULAR  point1  point2  point3 ,
        ELLIPTICAL  point  x_radius  y_radius start_angle  end_angle } ;
```

Arguments:

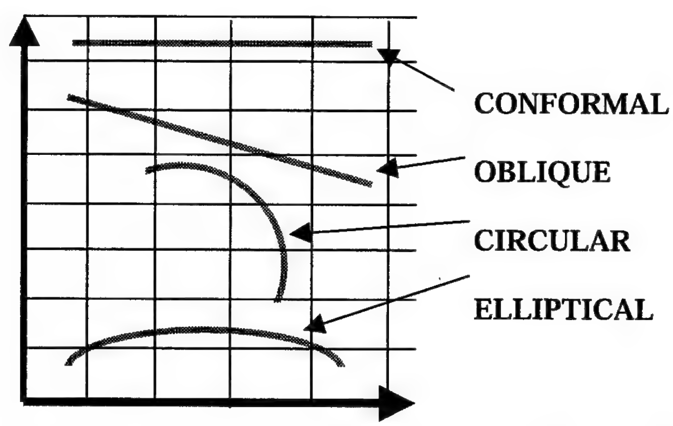
line	- name of spatial line, user-defined.
point	- spatial coordinates or name of spatial point defined in POINT command.
x_radius	- radius in x-axis (m).
y_radius	- radius in y-axis (m).
start_angle	- end-point angle (rad).
end_angle	- end-point angle (rad).

Description:

The LINE command is used to specify the locations of lines in space. Lines are created by entering the names of spatial points defined in POINT commands or by entering point coordinates. In other commands that require a line, you can enter the name of any line previously defined in a LINE command.

The line name provides a unique label for the line. If a line name is duplicated in another LINE command, the original point data will be lost, and an error message will result. If desired, variable substitution may be used to "subscript" similar spatial lines (see Section 4.4).

There are four shape options: CONFORMAL, OBLIQUE, CIRCULAR, and ELLIPTICAL. (The CIRCULAR and ELLIPTICAL shapes are restricted to 2D simulations.) See the following figure for the shapes.



This figure illustrates the four line shapes available in MAGIC.

All line shapes may be described by a metric of the parametric path length s.

$x_i = x_i^{initial} + m_i(s)\ s$, where $0 \le s \le 1$.

The nature of the metric m defines the line shape.

For many applications, it is desirable for the lines to "bend" to conform to the local coordinates. This is easily accomplished using the CONFORMAL option. If more than two points are entered, then more line segments will be added in a "connect-the-dots" manner. Thus, three end points will produce two line segments, etc., and the segments will be joined end-to-end, continuously. For example, a pie-shaped outline can easily be traced out in polar coordinates using this option. Each line segment must be conformal. Together, they comprise a single line identified by the line name.

The simplest shape option is CONFORMAL, which requires only the specification of the end points. The metric for a conformal line segment requires that one of the m_i's is a nonzero constant, and that the remaining m_i's are identically zero.

The OBLIQUE shape option requires only the specification of two end points. This will produce a single oblique line, irrespective of the coordinate system (which serves only to define the points). The metric for an oblique line segment requires only that the metric coefficients be constants and that at least one of them is nonzero.

The CIRCULAR option (2D simulations only) will produce a circular arc, irrespective of the coordinate system. The points required are first, point1, which locates the center of the circle, followed by the arc end-point locations. The convention is that the arc is always drawn counterclockwise, from point2 to point3. This option requires that the user maintain the correct radius to within a reasonable approximation.

The ELLIPTICAL option (2D simulations only) will produce an elliptical arc, irrespective of the coordinate system specified. The point required is the center point, which locates the center of the ellipse. Next, the magnitudes of the axes, x_radius and y_radius, are entered. (Note that the orientation of the ellipse is limited to coincide with the major axes.) Finally, two angles specify the end-points of the arc. The convention is that the arc is always drawn counterclockwise, from the start_angle to the end_angle. (The angles are measured counter-clockwise from the x-axis.)

Restrictions:

1. No two lines can have the same line name; if a line name is duplicated, the original spatial data will be lost, and an error message will be produced.

2. Conformal lines must be conformal in one coordinate for 2D and in two coordinates for 3D. For example, in 3D polar coordinates, an arc at constant radius and constant axial location is allowed, whereas tracing a helical path is not, since a helix is conformal only in radius.

3. The CIRCULAR and ELLIPTICAL shapes are restricted to 2D simulations.

See Also:

SYSTEM, Ch. 10
POINT, Ch. 10
AREA, Ch. 10
VOLUME, Ch. 10
DELETE, Ch. 10
LIST, Ch. 10

Examples:

In the examples which follow, it is assumed that the cartesian coordinate system is active unless otherwise noted.

1. To enter a line representing the axis of symmetry (running from 0 to Zmax in z) in a three-dimensional polar (r,ϕ,z) coordinate system, some possible commands follow. Note that we could have selected the shape, CONFORMAL, and obtained the same line.

```
SYSTEM POLAR ;
POINT a 0,0,0 ;
POINT b 0,0,Zmax ;
LINE axis OBLIQUE a,b ;
```

2. To create the outline of a box of width, W, and height, H, at constant z = 0 with a single LINE command, use the continuation convention with the commands,

```
POINT a 0,0,0 ;
POINT b W,0,0 ;
POINT c W,H,0 ;
POINT d 0,H,0 ;
LINE  outline  OBLIQUE a,b,c,d,a ;
```

The result will be four line segments with common end-points. The four segments are part of a single line called "outline." Note that this is a line with multiple segments, and <u>not</u> an AREA.

3. This example uses DO / ENDDO commands to create ten lines at uniformly-spaced axial locations at constant angle, Phi, between a cathode at radius, Rc, and an anode at radius, Ra. Using variable substitution, the individual lines are given the line names, line1, line2, ..., line10.

```
Z  =  Z0 - dZ;
DO  I  1, 10 ;
   Z = Z + dZ ;
   POINT   a'i' Rc,Phi,Z ;
   POINT   b'i' Ra,Phi,Z ;
   LINE  line'i'  OBLIQUE  a'i' b'i' ;
ENDDO ;
```

These lines could be used to measure cathode-anode voltages along a coaxial cable, for example. Note that all points and lines have been given unique names, as required.

4. The following commands create a circular arc of radius, R, subtending half of a circle between 0 and pi using the CONFORMAL option.

```
SYSTEM  POLAR ;
POINT  a  R,0,Z ;
POINT  b  R,Pi,Z ;
LINE  arc  CONFORMAL  a  b ;
```

AREA Command

Function: Specifies an area in space.

Syntax:

```
AREA  area
        { CONFORMAL  point1  point2 ,
          RECTANGULAR  point1  point2 ,
          FUNCTIONAL  f(x1,x2,x3) ,
          POLYGONAL  point  point  point...,
          FILLET point1 point2 radius start_angle end_angle   [2D only]
          QUARTERROUND iquadrant point radius                 [2D only]
          SINUSOID {X1,X2} point1 point2 point3 } ;           [2D only]
```

Arguments:

area	- name of area object, user-defined.
point	- spatial coordinates or name of spatial point defined in POINT command.
f(x1,x2,x3)	- function of spatial coordinates, defined in FUNCTION command.
radius	- radius of surface (m).
angle	- angle in degrees.
iquadrant	- quadrant specification. =1, 2, 3, 4.

Description:

The AREA command is used to specify the location of an area in space. The area name provides a unique label for the area. If an area name is duplicated in another AREA command, the original area data will be lost, and an error message will be produced. If desired, variable substitution may be used to "subscript" similar spatial areas (see Section 4.4).

The surface associated with the area must be conformal with one coordinate. That is, areas of completely arbitrary orientation are not allowed. There are four shape options: CONFORMAL, RECTANGULAR, FUNCTIONAL and POLYGONAL.

For many applications, it is desirable for an area to "bend," so that the outline conforms to one of the coordinates. This is easily accomplished using the CONFORMAL option, which requires only the specification of two opposing corner points. This option works in exactly the same way as the RECTANGULAR option, except that the shape will be distorted to match the specified coordinate system. It will produce rectangular shapes in cartesian coordinates and is ideal for entering a pie-shaped area or the outer surface of a cylinder in polar coordinates, for example.

The simplest option is RECTANGULAR, which requires only the specification of two opposing corner points. The RECTANGULAR option is restricted to cartesian coordinates.

The FUNCTIONAL option can be used to create an area of arbitrary shape in a plane which is conformal with one coordinate. To utilize it, you enter the name of a function which you have previously specified in a FUNCTION command. The shape of the area is determined by the sign of the function, evaluated throughout the spatial coordinates. The area is defined wherever the sign is negative, but does not exist wherever the function is positive or vanishing (zero). Repeating, or periodic, structures are particularly simple to create with this option.

The POLYGONAL option can be used to create an area bounded by straight lines connecting the corner points. Simply enter the names of the corner points, in order. Lines are not allowed to intersect (cross), the perimeter must be continuous, and the first and last corner point must be the same to provide closure.

The FILLET option can be used to create an area in a bounding rectangle. The bounding rectangle is always conformal to the x1 and x2 coordinate directions. For the fillet definition, the first argument, point 1 is the center point, as can be seen in the following figure. The second point, point 2, is the extremum of the bounding rectangle. The relative locations of these points determine the quadrant placement of the fillet area. The argument, radius, determines the circular cut in the rectangular region. The angles, θ_i, and θ_f, are the starting and ending angles in degrees measured from the positive direction x1-coordinate. To obtain a good match between the grid and the fillet area, MARK the FILLET area, specifying only a size parameter. The marking algorithm will simultaneously mark both the x1 and x2 coordinates in order to maintain the best match with the area.

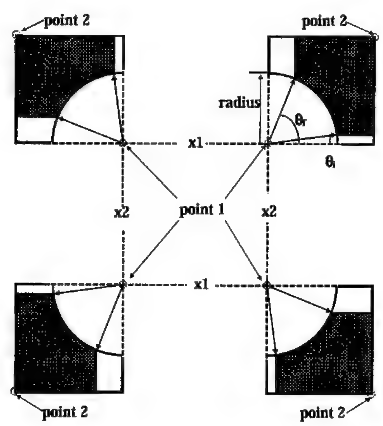


Figure illustrating the arguments associated with the FILLET area shape.

The QUARTERROUND option can be used to create a pie-shaped area that is 1/4 of a circle. This command would typically be used in cartesian or cylindrical coordinates. The radial edges of the pie shape are always conformal to the x1 and x2 coordinate directions. The first argument denotes the relative quadrant (with respect to the center point, in which the quarterround area is to be placed. The second argument, point 1, is the center point, as can be seen in the following figure. The argument, radius, determines the circular arc radius. To obtain a good match between the grid and the fillet area, MARK the area, specifying only a size parameter. The marking algorithm will simultaneously mark both the x1 and x2 coordinates in order to maintain the best match with the area.

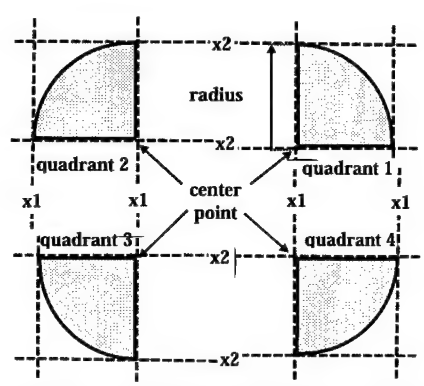


Figure illustrating the arguments associated with the QUARTERROUND shape.

The SINUSOIDAL option can be used to create an area with one of the surfaces having a sinusoidal ripple. The bounding rectangle is always conformal to the x1 and x2 coordinate directions. The area is defined by three points and a direction alignment (x1 or x2) which indicates the coordinate axis along which the wavelength is defined. The arguments, point 1, point 2, and point, are located as indicated in the following figure. The first two points determine the depth and wavelength of the sinusoidal surface variation. The third point, point 3, is the extremum of the bounding rectangle. The relative locations of these points determine the quadrant placement of the fillet area. To obtain a good match between the grid and the fillet area, MARK the area, specifying only a size parameter. The marking algorithm will simultaneously mark both the x1 and x2 coordinates in order to maintain the best match with the area. The grid along the wavelength direction will be uniform.

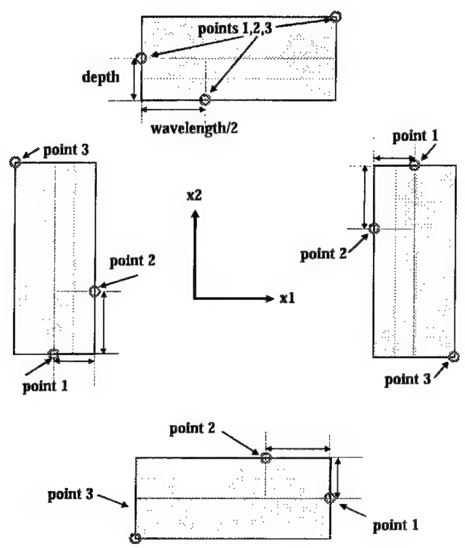


Figure illustrating arguments associated with the SINUSOIDAL shape.

Restrictions:

1. No two areas can have the same area name; if an area name is duplicated, the original spatial data will be lost, and an error message will be produced.

2. All areas are required to have surfaces conformal with one coordinate (x3 in 2D simulations).

3. The RECTANGULAR option is available only in cartesian coordinates.

4. In the POLYGONAL option, lines are not allowed to intersect (cross), the perimeter must be continuous, and the first and last corner point must be the same to provide closure.

5. The area shapes QUARTERROUND, FILLET, and SINUSOIDAL are only available in 2D.

See Also: **SYSTEM, Ch. 10**
POINT, Ch. 10
LINE, Ch. 10

VOLUME, Ch. 10
DELETE, Ch. 10
LIST, Ch. 10

Examples:

In the examples which follow, it is assumed that the cartesian coordinate system is active unless otherwise noted.

1. To create a rectangular area of width, W, and height, H, conformal with z and with one corner at the origin, you can use the following commands:

```
POINT a 0,0,0 ;
POINT b W,H,0 ;
AREA  BOX  RECTANGULAR  a,b  ;
```

An equivalent alternative (in cartesian coordinates) is the following command.

```
AREA  BOX  CONFORMAL  a,b  ;
```

2. To create an area in polar coordinates between radii Ra and Rb which subtends angle zero to pi at constant z, you can use the following commands.

```
SYSTEM  POLAR  ;
POINT a Ra,0,z ;
POINT b Rb,Pi,z ;
AREA  Pie_Shape  CONFORMAL  a,b ;
```

3. This example uses the FUNCTIONAL option to create a circular area of radius, R, about the origin at constant z in cartesian coordinates.

```
FUNCTION  alfa (x,y,z) =  (+x*x+y*y-R*R)
            * STEP(z,0)*STEP(0,z)  ;
AREA  beta  FUNCTIONAL  alfa  ;
```

The function called alfa is clearly negative only within a circle of radius, R. The created area will be named "beta."

4. The following commands create and mark a QUARTERROUND area in the first quadrant.

```
GRAPHICS PAUSE  ;
SYSTEM CARTESIAN ;
AREA QUAD1 QUARTERROUND 1 -1 -1 50CM ;
MARK QUAD1 SIZE 3CM ;
DISPLAY_2D;
AUTOGRID ;
CONDUCTOR QUAD1 ;
DISPLAY_2D PERIMETER GRID ;
```

```
START ;
STOP;
```

5. The following commands create and mark a FILLET area symmetrically placed around the origin. One fillet object is generated in each quadrant orientation.

```
GRAPHICS PAUSE  ;
SYSTEM CARTESIAN ;
AREA QUAD1 FILLET  -1 -1 -25CM -25CM  50CM 0 90 ;
MARK QUAD1 SIZE 3CM ;
AREA QUAD2 FILLET  +1 -1 25CM -25CM   50CM 90 180 ;
MARK QUAD2 SIZE 3CM ;
AREA QUAD3 FILLET  1 1 25CM 25CM  50CM 180 270;
MARK QUAD3 SIZE 3CM ;
AREA QUAD4 FILLET -1 +1 -25CM 25CM 50CM 270 360 ;
MARK QUAD4 SIZE 3CM ;
AUTOGRID ;
DISPLAY_2D ;
CONDUCTOR QUAD1 ;
CONDUCTOR QUAD2 ;
CONDUCTOR QUAD3 ;
CONDUCTOR QUAD4 ;
DISPLAY_2D PERIMETER GRID ;
START ;
STOP;
```

6. The following commands create and mark a SINUSOIDAL area. AUTOGRID is used to create a grid and then the CONDUCTOR command is used to set the area to the conducting property.

```
IQUADRANT = 4 ;
NRESOLU = 14;
GRAPHICS PAUSE  ;
SYSTEM CARTESIAN ;
XN_WAVELENGTHS = 2.5 ;
PHASE = -1 ;
WAVELENGTH = 1M ;
DELTA_WAVE = WAVELENGTH/NRESOLU ;
DEPTH = 25CM ;
SUPPORT_DEPTH = 25CM ;
XA = 0.0 ;
XB = XA+0.5*WAVELENGTH ;
XC = XA+WAVELENGTH*XN_WAVELENGTHS ;
YA = PHASE*0.5*DEPTH  ;
YB = YA - PHASE*DEPTH ;
YC = MAX(YA,YB)+SUPPORT_DEPTH ;
IF (IQUADRANT.EQ.1) THEN ;
     AREA QUAD SINUSOID X1 XA,YA XB,YB XC,YC ;
ENDIF ;
IF (IQUADRANT.EQ.2) THEN ;
```

```
     AREA QUAD SINUSOID X2  -YA,XA -YB,XB -YC,XC ;
ENDIF ;
IF (IQUADRANT.EQ.3) THEN ;
     AREA QUAD SINUSOID X1 -XA,-YA -XB,-YB -XC,-YC ;
ENDIF ;
IF (IQUADRANT.EQ.4) THEN ;
     AREA QUAD SINUSOID X2  +YA,-XA +YB,-XB +YC,-XC ;
ENDIF;
MARK QUAD SIZE DELTA_WAVE ;
AUTOGRID ;
DISPLAY_2D ;
CONDUCTOR QUAD ;
DISPLAY_2D PERIMETER GRID ;
START ;
STOP;
```

VOLUME Command

Function: Specifies a volume in space (3D simulations only).

Syntax:

```
VOLUME  volume
        { CONFORMAL  point1  point2 ,
          PARALLELPIPEDAL point0  point1  point2  point3 ,
          CYLINDRICAL  point1  point2  radius ,
          ANNULAR  point1  point2  radius  radius ,
          SPHERICAL  point  radius ;
          FUNCTIONAL  f(x1,x2,x3)  [ point1  point2 ] ,
          EXTRUDED  area  line } ;
```

Arguments:

volume	-	name of volume object, user-defined.
point	-	coordinates of spatial point or name defined in POINT command.
radius	-	radius of cylinder or sphere (m).
f	-	signed function of spatial coordinates, defined in FUNCTION command.
area	-	name of area object, defined in AREA command.
line	-	name of line object, defined in LINE command.

Description:

The VOLUME command is used to specify the location of a volume in space. It is used for 3D simulations only.

The volume name provides a unique label for the volume. If a volume name is duplicated in another VOLUME command, the original point data will be lost, and an error message will be produced. If desired, variable substitution may be used to "subscript" similar spatial volumes (see Section 4.4).

There are seven shape options: CONFORMAL, PARALLELEPIPEDAL, CYLINDRICAL, ANNULAR, SPHERICAL, FUNCTIONAL, and EXTRUDED.

The CONFORMAL option produces a volume with six sides, each of which is conformal to a coordinate in the active coordinate system. In cartesian coordinates, it produces a rectangular shape. In polar coordinates, it produces a section of an annulus. In spherical coordinates, it produces a section of a spherical shell. Specify two opposing corner points of the conformal volume to create this shape. The first point must be at the lowest x1,x2,x3 coordinates, and the second point must be at the highest x1,x2,x3 coordinates.

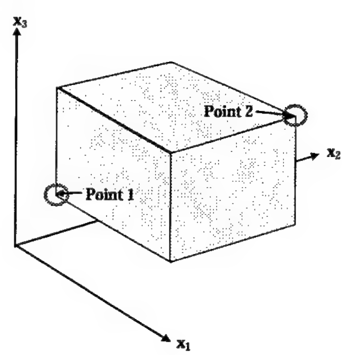


Figure illustrating CONFORMAL volume and argument definition.

The PARALLELPIPEDAL option will produce a parallelepiped of arbitrary orientation and with flat surfaces, irrespective of the coordinate system. Enter any one corner point (labeled 0), followed by the only three corner points which are immediately adjacent to the first.

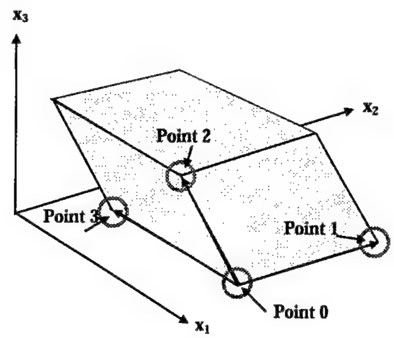


Figure illustrating PARALLELPIPEDAL volume and argument definition.

The CYLINDRICAL option will produce a cylindrical shape of arbitrary orientation, irrespective of the coordinate system selected. Specify the two end points of the cylinder axis, followed by the cylinder radius.

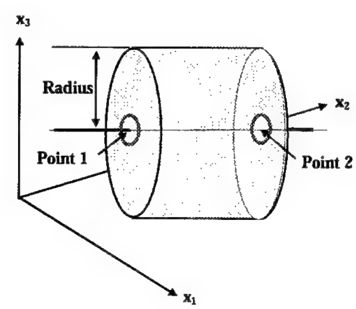


Figure illustrating CYLINDRICAL volume and argument definition.

The ANNULAR option will produce a hollow cylinder. Specify the two end points of the cylinder axis, followed by two radii (inner and outer).

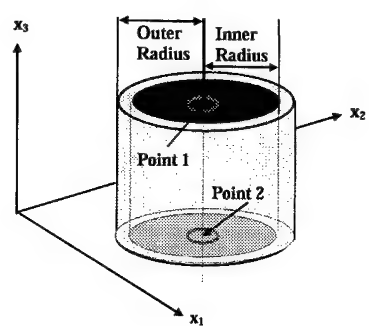


Figure illustrating ANNULAR volume and argument definition.

The SPHERICAL option will produce a spherical shape, irrespective of the coordinate system selected. Specify the center point of the sphere and the sphere radius.

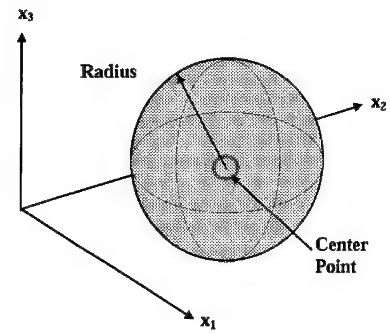


Figure illustrating SPHERICAL volume and argument definition.

The FUNCTIONAL option is used to create an arbitrarily shape, making use of a volume function, f. Points at which the value of f is less than zero lie within the volume. Points at which the value of f is greater than or equal to zero lie outside the volume. Only points within the clip box volume specified by the (optional) corner points are included within the volume.

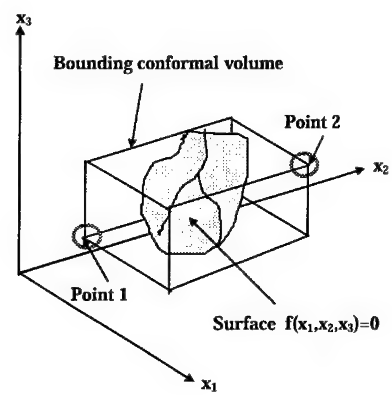


Figure illustrating the FUNCTIONAL volume and argument definition.

The EXTRUDED option is used to generate a shape of uniform cross section by extruding an area over a path defined by a line. The area in this option must be conformal in one of the three coordinates. In addition, the only area shapes allowed are CONFORMAL and POLYGONAL. The extrude line must have a component perpendicular to the plane of the area and may be of the shape type CONFORMAL or STRAIGHT. A good method of using the EXTRUDED option is to have the first point of the area definition match the starting point of the extrude line.

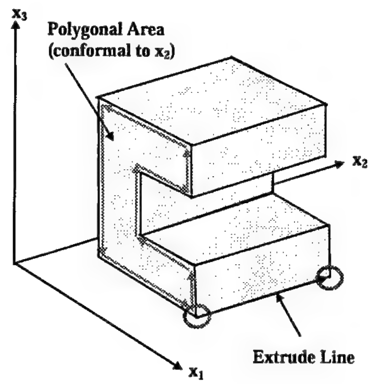


Figure illustrating the EXTRUDED volume and argument definition.

Restrictions:

1. No two volumes can have the same name; if a volume name is duplicated in another VOLUME command, the original data will be lost, and an error message will be produced.

2. All six surfaces of a conformal volume will be conformal.

3. For the EXTRUDED option, the surface associated with the specified area must be conformal with one coordinate. The line of extrusion cannot be parallel to this surface.

See Also: **SYSTEM, Ch. 10**
POINT, Ch. 10
LINE, Ch. 10
AREA, Ch. 10
DELETE, Ch. 10
LIST, Ch. 10

Examples:

1. The following commands specify a perfect cube two meters on a side using the CONFORMAL option.

```
SYSTEM CARTESIAN
POINT  a  2,5,10 ;
POINT  b  4,7,12 ;
VOLUME cube CONFORMAL a,b ;
VOLUME cube CONFORMAL  2,5,10  4,7,12 ;
```

2. The following command specifies a cylindrical volume with a base point at (2,2,2), the top point at (4,5,6), and a radius of 8.

```
POINT  a  2,2,2 ;
POINT  b  4,5,6 ;
VOLUME post CYLINDER a,b  8 ;
```

3. The following commands specify spherical volumes using three alternative methods. The center of the sphere is at (1,2,3), and the radius of the sphere is four meters.

```
POINT  center  1,2,3 ;
VOLUME ball1 SPHERICAL center 4 ;
FUNCTION ball(x,y,z)=SQRT((x-1)**2+(y-2)**2+(z-3)**2)-4**2;
VOLUME ball2 FUNCTIONAL ball ;
POINT octant 10,10,10 ;
VOLUME ball3 FUNCTIONAL ball center octant ;
```

Note that the third case (ball3) uses the (optional) clip box to restrict the object to the positive, definite octant of the spherical volume, so the commands above would produce two identical spheres and one spherical octant.

4. The following commands generate an extruded shape named "Extrusion." The cross section is a three-sided, polygonal-shaped area named "Wedge," with corner-points labeled A, B, and C. (Note that point A must repeat to enclose the area.) The extruded length is defined by the line named "Path," which runs from point A to point D.

```
AREA  Wedge  POLYGONAL  A B C A ;
LINE  Path  CONFORMAL  A  D ;
VOLUME  EXTRUDED  Wedge  Path ;
```

DELETE Command

Function: Deletes previously entered spatial objects or markers.

Syntax: DELETE { OBJECT, MARK } object ;

Arguments: object - name of existing spatial object, user-defined in POINT, LINE, AREA, or VOLUME commands.

Description:

Note: The DELETE command is presently inoperable.

The DELETE command is used to remove specified objects or markers from an input file. Typically, this file is read in or created in the interactive mode. Note that DELETE MARK deletes only the markers, and not the object itself. DELETE OBJECT will delete both the object and any markers associated with it. Use of a specific object name allows objects or markers to be deleted.

See Also: **POINT, Ch. 10**
LINE, Ch. 10
AREA, Ch. 10
VOLUME, Ch. 10
LIST, Ch. 10
MARK, Ch. 11

Examples:

We shall create two points, labeled a and b, and then delete b and create c.

```
POINT  a   Xa,Ya,Za  ;
POINT  b   Xb,Yb,Zb  ;
DELETE  OBJECT  b  ;
POINT  c  Xc,Yc,Zc  ;
```

The final result is two points, labeled a and c.

LIST Command

Function: Lists previously entered spatial objects and markers.

Syntax:

LIST [{ object, POINT, LINE, AREA, VOLUME }] ;

Arguments:

object - name of spatial object, user-defined in POINT, LINE, AREA, or VOLUME commands.

Description:

The LIST command is used to list specified objects or types of objects (POINT, etc.) and associated markers from an input file. Typically, this file is read in or created in the interactive mode. Use of a specific object name or object type restricts the list to specific objects or types of objects; otherwise, all objects and markers will be listed.

See Also: **POINT, Ch. 10**
LINE, Ch. 10
AREA, Ch. 10
VOLUME, Ch. 10
DELETE, Ch. 10
MARK, Ch. 11

Examples:

(1) We shall create two points, labeled a and b, and then list all and create c.

```
POINT  a   Xa,Ya,Za  ;
POINT  b   Xb,Yb,Zb  ;
LIST  ;
POINT  c  Xc,Yc,Zc  ;
```

The list will contain the two points labeled a and b, but not c.

(2) We next mark point c and list.

```
MARK  c  ;
LIST  c  ;
```

Marker locations will be listed for point c, but not for points a and b, which were not marked.

11. GRIDS

This Section covers the following commands:

DURATION
TIMER
MARK
AUTOGRID
GRID ORIGIN
GRID EXPLICIT
GRID UNIFORM
GRID QUADRATIC
GRID PADE

You can use these commands to create grids in time and space. The DURATION command is used to specify the time span covered by the simulation. The TIMER command is used to specify trigger times for use by other commands, e.g., to specify when certain measurements are to be made.

To construct a spatial grid in any of the three coordinates (x_1, x_2, or x_3), you have two choices: the AUTOGRID command (automatic), and the GRID commands (manual). The AUTOGRID command is used in conjunction with MARK commands to generate the grid automatically. The MARK command sets markers and cell sizes at key locations on spatial objects (see Ch. 10). The AUTOGRID command uses this data to generate the grid. If you use the MARK and AUTOGRID commands, no other commands are necessary to construct a spatial grid.

You can also construct a spatial grid manually using GRID commands. The GRID ORIGIN command fixes the value of the coordinate at the origin. As an option, you may also enter the cell size at the origin. Use the other GRID commands to construct consecutive sections of the grid. The different options (EXPLICIT, UNIFORM, QUADRATIC, etc.) allow different functional variations for the cell size within each section. You can use any number and combination of GRID commands. The first command builds the first section, beginning from the origin. Each subsequent GRID command extends the grid by adding a new section of grid that is appended to the previous section.

In 2D simulations, the spatial grid index extends over the range

$2 \leq i \leq ixmax$,

and in 3D the range is

$1 \leq i \leq ixmax$.

Usually, ixmax is chosen to meet the resolution requirements (subject to code limitations). To facilitate grid extension, the code automatically generates and updates a system variable, ISYS$InMX (n = 1, 2, 3), containing the current number of defined grid points. This variable may be referenced in other commands. Other system variables generated by the AUTOGRID and GRID commands include SYS$XnMN and SYS$XnMX, the initial and final full-grid points.

DURATION Command

Function: Specifies the time span for a time-dependent simulation.

Syntax: DURATION time_span ;

Arguments: time_span - simulation time span (sec.).

Description:

The DURATION command is used to specify the time_span for a time-dependent simulation.

Two system variables are produced when this command is entered. The variable, SYS$RUNTIME, is set equal to time_span. The other variable, ISYS$KMAX, representing the total number of electromagnetic time steps in time_span, is calculated. Both of these system variables can be used in subsequent commands.

See Also: **MAXWELL, Ch. 17**

TIMER Command

Function: Defines a time trigger.

Syntax:
```
TIMER timer PERIODIC { INTEGER, REAL } start_time [ stop_time [ time_increment ] ]
        [ INTEGRATE time_interval ] ;
```

(or)
```
TIMER timer DISCRETE { INTEGER, REAL } trigger_time1 [ trigger_time2, ... ]
        [ INTEGRATE time_interval ] ;
```

Arguments:

timer	- timer name, user-defined.
start_time	- do-loop start time or time index.
stop_time	- do-loop stop time or time index (default = infinity).
time_increment	- do-loop time or time index increment (default = MAX(1,start_time)).
trigger_time1,...	- discrete values of time or time indices.
time_interval	- time or time-index interval for integration.

Defaults:

In addition to any timers which you create explicitly, there are several timers created by the system for various purposes, all of which are available for your use. One class of these is used only in conjunction with the EIGENMODE algorithm. Since there is no "time" variable for eigenmode calculations, these timers trigger when certain events occur:

TSYS$EIGENMODE	- triggers when converged eigenmode is found
TSYS$EIGEN	- triggers periodically during convergence process
TSYS$EIGFIRST	- triggers on initiating new eigenmode search

System timers used in time-dependent simulations include the following:

TSYS$FIRST	- triggers at the beginning of the simulation
TSYS$LAST	- triggers at the completion of the simulation
TSYS$IMPORT	- triggers to import particles and fields
TSYS$LORENTZ	- triggers for particle kinematics (see LORENTZ TIMING option)
TSYS$OBSERVE	- triggers to observe variables (see OBSERVE INTERVAL option)

Description:

Many commands require instruction as to when they are to be exercised, or "triggered." A simple example is the RANGE command, which produces a plot only when it is exercised. The TIMER command defines a time trigger, which is just a sequence of times. Once it is defined, a time trigger can be applied to any command requiring one (such as RANGE), simply by entering the timer name.

There are two options, PERIODIC to enter a periodic trigger, and DISCRETE to enter an irregular trigger. With either one, you can specify REAL to enter times in seconds or INTEGER to enter time indices. (Time indices are used with the time_step (TIME_STEP, Ch. 17) to compute times in seconds.) The PERIODIC option causes a do-loop to calculate trigger_times. A data entry is always required for start_time (which must be positive), but the data entries for stop_time and time_increment are optional. If data is not entered, then stop_time is set to infinity, and time_increment is set to unity or start_time, whichever is greater. The

DISCRETE option allows the trigger_times to be entered explicitly. This option is useful when the trigger_times are few or irregularly spaced. The trigger_times must be entered in increasing order.

Many commands which require timers make physical measurements of fields, currents, etc. In such applications, the INTEGRATE option in TIMER will cause integration to be performed over a time_interval before the trigger_time. When the measurement is initiated (at trigger_time - time_interval), it will be made continuously (every time step) for the period of time_interval. Upon completion (trigger_time), the measurements cease, and the average value is calculated and output (see Examples, below).

Restrictions:

1. A single TIMER command with the DISCRETE option allows up to ten trigger_times to be entered.

2. In using the INTEGRATION option, you cannot allow measurements initiated by two trigger_times to overlap. In other words, the time_interval must not exceed the difference between the trigger_times.

3. The TIMER command should not be used when employing the EIGENMODE command. Use the predefined timers instead.

See Also: **TIME_STEP, Ch. 17**
EIGENMODE, Ch. 19
LORENTZ, Ch. 18
OBSERVE, Ch. 22
RANGE, Ch. 23, and other output commands which require timers.

Examples:

1. The following example requests TABLE output of fields E1, B2, and B3 in an area named "Area" every 20 time steps, starting on time index 10 and continuing through time index 90. (Printout will occur on time indices 10, 30, 50, 70, and 90. Printout will not occur on time index 100.)

```
TIMER  P_timer PERIODIC INTEGER 10,100,20 ;
TABLE  E1 Area P_timer ;
TABLE  B2 Area P_timer ;
TABLE  B3 Area P_timer ;
```

2. The following example illustrates use of the INTEGRATE option. The measurement involves time-averaged gain as a function of axial position in an RF amplifier. This complex diagnostic is obtained using only three commands:

```
TIMER Gain_timer PERIODIC INTEGER K_plot, Infinity, K_plot
     INTEGRATE K_period  ;
FUNCTION Gain(P_Poynting) = 10.0 * LOG10 (P_Poynting / P_input ) ;
RANGE  POYNTING  P1  Input_port  Gain_timer  TRANSVERSE_INTEGRAL  path
     TRANSFORM  Gain  ;
```

Here, P_input is the input power, K_plot is the time_increment (time between plots), and K_period is the time_interval for integration (number of time steps in an RF period). Note that time_interval must be less than, or equal to, time_increment to prevent the integration measurements from overlapping and producing an error.

MARK Command

Function: Marks locations on spatial objects for automatic gridding.

Syntax:

MARK object [{ X1, X2, X3 }
[MINIMUM] [MIDPOINT] [MAXIMUM] [SIZE cell_size]] ;

Arguments:

object - name of spatial object, defined in POINT, LINE, AREA, or VOLUME command.

cell_size - cell size at marked location (m or rad).

Description:

Spatial objects may be created either before or after the spatial grid is generated. However, there is an advantage in defining the objects first, because this information can be used to generate the grid automatically. To do this, use the MARK command to mark key locations on a spatial object. The AUTOGRID command will then generate grid lines which conform to the object at the marked locations. Everywhere else, the (unmarked) spatial objects will be slightly distorted and/or shifted to conform to the grid. Thus, the MARK command can be used to preserve the exact size and shape of critical geometric features, such as a gap width, a conducting surface, a voltage integral, etc. Any type of spatial object (points, lines, areas, and volumes) may be marked using this command.

The arguments and options are as follows. The object name must be previously defined in a POINT, LINE, AREA, or VOLUME command. If you simply mark the spatial object (with no options), the result will be two marked locations in each coordinate, representing the extrema in the selected coordinate system. Thus, the number of (non-redundant) markers obtained with this option depends upon the spatial object and the number of coordinates, as indicated in the following table.

Spatial Object	2D Simulations	3D Simulations
point	2	3
line	4	6
area	4	5 (conformal)
volume	N. A.	6

You can refine this by using the optional arguments. First, select the coordinate (X1, X2, or X3). Next, you may identify critical locations, with the choices being MINIMUM, MIDPOINT, and MAXIMUM (or just MIN, MID, and MAX — MINIMUM and MAXIMUM refer to the extrema of the object, and MIDPOINT is their average value). You may choose none (blank) or any or all of these. If you do not specify, only the extrema will be marked. Finally, you may also specify the cell_size, which will be applied to all locations marked with that command. For very complex structures, you can always define spatial points at critical locations and mark those.

Note that, under certain circumstances, marked locations may be merged, shifted, or even eliminated. This can occur when two or more marked locations are very close together (compared with the cell_size). When the AUTOGRID command is processed, all marked locations are ranked in order, and a complex algorithm operates to resolve any problems resulting from under- or over-specification. For example, any exact duplicates are eliminated, since they would produce a cell of zero width. Other, more complicated tests ensure that the actual spatial grid does not fall below the minimum cell_size, which would adversely impact simulation run time

due to the Courant stability criterion. It is good practice to display the spatial grid and to compare positions of the actual grid lines with the original marked locations, shown by arrowheads (DISPLAY, Ch. 24).

Restrictions:

1. A spatial object must be created before it can be marked.

2. The MARK commands should precede the (grid generation) AUTOGRID command. Any subsequent MARK commands will be ignored.

3. Be aware that marked locations may be merged, shifted, or even eliminated by the AUTOGRID algorithm.

4. Do not mark the third coordinate in a 2D simulation.

5. Objects created using the FUNCTIONAL option cannot be marked directly. Instead, create points at critical locations on the surface and mark those.

See Also:
SYSTEM, Ch. 10
POINT, Ch. 10
LINE, Ch. 10
AREA, Ch. 10
VOLUME, Ch. 10
DELETE, Ch. 10
LIST, Ch. 10
AUTOGRID, Ch. 11
DISPLAY, Ch. 24

Examples:

(1) To mark a point named "a" in both coordinates (2D simulation), the command is simply

```
MARK  a ;
```

This would preserve this location in the grid to facilitate a precise field measurement, for example. (Note that this simple form of the MARK command always produces two marked locations in each coordinate, even for a point. However, they will appear as a single marker in each coordinate.)

(2) To mark the minimum x2 location of an object named "anode," the command is

```
MARK anode X2 MINIMUM ;
```

(3) If we wanted to mark the minimum and maximum in x1 and to specify a cell_size of 0.0005 meters at both locations, the command could read

```
MARK anode X1 MIN MAX SIZE 0.0005  ;
```

or just

```
MARK anode X1 SIZE 0.0005  ;
```

(4) To delete all marked locations on the anode, use the command

```
DELETE MARK anode  ;
```

(5) To list all of the marked locations, enter the command

```
LIST  ;
```

From the examples above, only the point "a" marked locations would be listed, since those for the "anode" were deleted in Example 4.

AUTOGRID Command

Function: Generates the spatial grid automatically from spatial markers.

Syntax: AUTOGRID [{ X1, X2, X3 } [total_cells]] ;

Arguments: total_cells - approximate number of cells desired (default = 100).

Description:

The AUTOGRID command is used to generate the spatial grid automatically. It makes use of previously-defined spatial markers (see MARK, Ch. 11). If the AUTOGRID and MARK commands are used, no other gridding commands are needed.

There are few options in AUTOGRID, since the spatial markers control most of the gridding logic. You may choose to grid one coordinate automatically (e.g., AUTOGRID X1) and the other(s) manually with GRID commands. Finally, if the spatial marker data is sparse, you may wish to enter the desired number of total_cells in that coordinate.

The AUTOGRID algorithm may merge, shift, or even eliminate some marked locations to resolve problems of over-specification. However, the algorithm always places grid lines precisely on all of the remaining markers. Between two adjacent markers, the cell size always obeys a Pade equation (see GRID PADE, Ch. 11). In general, the cell sizes will vary. Under certain circumstances, a uniform grid can result.

The algorithm logic depends upon the data that is available to it. To grid a region between two markers, the Pade prescription requires exactly three parameters, e.g., the distance between markers and the cell size at each marker. The algorithm calculates the distances between markers, and it uses the following approach to obtain cell sizes.

First, if the MARK commands have specified cell sizes at all of the markers, no more is required. If an interior marker cell size is missing, it will be calculated from available cell sizes by linear interpolation. If an exterior marker cell size is missing, it will be set equal to the nearest cell size (no extrapolation). If no cell sizes have been entered, then the total_cells and the total distance will be used to calculate the marker cell sizes.

Once all marker cell sizes have been calculated, they are adjusted to ensure an integer number of cells between markers. Finally, the grid itself is computed from Pade equations.

Restrictions:

1. There is an upper limit to the number of spatial grid points in each coordinate.

2. There is a separate upper limit to the product in all coordinates, i.e., the total number of cells.

3. No grid should be entered for the third coordinate (X3) in a 2D simulation.

See Also: **MARK, Ch. 11**

GRID ORIGIN Command

Function: Creates an origin in the specified coordinate axis.

Syntax: GRID ORIGIN { X1, X2, X3 }
axis_origin [cell_size [grid_index]] ;

Arguments:

axis_origin - coordinate value in meters or radians (default = 0.0).
cell_size - cell size at axis_origin in meters or radians.
grid_index - optional grid index at axis_origin, integer, default = 1 (3D) or 2 (2D).

Description:

The GRID ORIGIN command is used to specify the origin in one of the coordinate axes. It should be entered before any of the GRID options are used to build sections of the grid.

If the axis_origin is not specified in a GRID command, the lowest value spatial marker is used. If there are no spatial markers, a value of zero will be used. The cell_size at the axis_origin can also be specified. If so, it may be used in building the first section of grid. The optional specification of grid_index provides control over the simulation position in index space; normally, it is not needed.

Restrictions:

No grid should be entered for the third coordinate (X3) in a 2D simulation.

See Also:

GRID EXPLICIT, Ch. 11
GRID UNIFORM, Ch. 11
GRID QUADRATIC, Ch. 11
GRID PADE, Ch. 11
GRID SIN / COS, Ch. 11

Examples:

A simulation of a coaxial cable in polar coordinates requires a grid between the inner and outer radii (R_inner and R_outer), but there is no reason to grid the region between the center axis and R_inner, since it is not used. Therefore, we can set the radial axis origin to R_inner with the command

```
GRID  ORIGIN  X2  R_inner  ;
```

GRID EXPLICIT Command

Function: Creates an arbitrarily varying grid in a region.

Syntax: GRID EXPLICIT { X1, X2, X3 } CELLS cells [SIZE cell_size, ...] [GRID full_grid, ...] ;

Arguments:

cells	-	number of cells in the region.
cell_size	-	cell size in meters or radians.
full_grid	-	grid values in meters or radians.

Description:

The EXPLICIT option allows an explicit sequence of data to be entered to create an arbitrary grid. The number of cells must be specified; then two options are available to specify the cell-by-cell data. The SIZE option allows a sequence of cell_size values to be entered. They are used internally to construct the grid. The GRID option allows full_grid values to be entered.

Restrictions:

1. There is an upper limit to the number of spatial grid points in each coordinate.

2. There is a separate upper limit to the product in all coordinates, i.e., the total number of cells.

3. No grid should be entered for the third coordinate (X3) in a 2D simulation.

See Also:

GRID ORIGIN, Ch. 11
GRID UNIFORM, Ch. 11
GRID QUADRATIC, Ch. 11
GRID PADE, Ch. 11
GRID SIN / COS, Ch. 11

GRID UNIFORM Command

Function: Creates a uniform grid in a region.

Syntax:

```
GRID UNIFORM { X1, X2, X3 }
        [ DISTANCE  region_size ]
        [ FIRST { MATCH, cell_size } ]
        [ CELLS cells ] ;
```

Arguments:

region_size	- length of the region in meters or radians.
cell_size	- cell size of region in meters or radians.
cells	- number of cells in the region.

Description:

The UNIFORM option results in equally-spaced full-grid points given by

xif = x1f + (i-1) cell_size

where cell_size can be entered directly or will be calculated from other parameters. Although a uniform grid can also be achieved using all of the other options, the UNIFORM option is the simplest way, and it reduces round-off errors in grid spacing.

The UNIFORM algorithm requires only two parameters to specify the grid completely. As indicated in the syntax, there are three options, and you must select two. If the parameters are either under- or over-specified, an error condition will result. The options are DISTANCE, FIRST, and CELLS. (The FIRST option refers to the first cell. If MATCH is entered, the algorithm will attempt to obtain the cell_size from the ORIGIN command or from the last cell in the preceding region. Otherwise, cell_size should be entered.)

If DISTANCE is one of the options specified, cell_size will be normalized to provide an exact integral number of cells, thus ensuring that grid lines fall precisely on the ends of the region_size.

Restrictions:

1. There is an upper limit to the number of spatial grid points in each coordinate.

2. There is a separate upper limit to the product in all coordinates, i.e., the total number of cells.

3. No grid should be entered for the third coordinate (X3) in a 2D simulation.

See Also:

GRID ORIGIN, Ch. 11
GRID EXPLICIT, Ch. 11
GRID QUADRATIC, Ch. 11
GRID PADE, Ch. 11
GRID SIN / COS, Ch. 11

Examples:

A simulation of a coaxial cable in cylindrical coordinates requires a grid between the inner and outer radii (R_inner and R_outer). We can set the radial origin to R_inner and divide the gap into 20 uniform cells with the commands

```
GRID  ORIGIN  X2  R_inner  ;
gap  =  R_outer  -  R_inner  ;
Ncells  =  20  ;
dR  =  gap  /  Ncells  ;
GRID  X2  UNIFORM  DISTANCE  gap  FIRST  dR  ;
```

Note that the DISTANCE and FIRST options are used to satisfy the requirement for two parameters.

GRID QUADRATIC Command

Function: Creates a quadratic grid in a region.

Syntax:

```
GRID QUADRATIC { X1, X2, X3 }
        [ DISTANCE region_size ]
        [ FIRST { MATCH, cell_size } ]
        [ LAST cell_size ]
        [ CELLS cells ] ;
```

Arguments:

region_size	- length of the region in meters or radians.
cell_size	- cell size in meters or radians.
cells	- number of cells in the region.

Description:

The FUNCTIONAL option results in a quadratically-spaced grid over a region. The ith full-grid point, xif, in a particular region is given by the formula,

$$x_i^f = x_1^f + (i - 1)\frac{(I^2 \delta x_1^f - d)}{I(I-1)} + (i-1)^2 \frac{(d - I\delta x_1^f)}{I(I-1)}, \quad 1 \le i \le I+1,$$

where d = the region_size, I = cells, and dx_1^f = cell_size for the last cell. This means that the cell_size is allowed to increase or decrease uniformly by a constant amount

$$\delta x_{i+1}^f = \delta x_i^f \pm \varepsilon,$$

where

$$\varepsilon = \frac{2(d - I\delta x_1^f)}{I(I-1)} .$$

The QUADRATIC algorithm requires only three parameters to specify the grid completely. As indicated in the syntax, there are four options, and you must select three. If the parameters are either under- or over-specified, an error condition will result. The options are DISTANCE, FIRST, LAST, and CELLS. (The FIRST option refers to the first cell. If MATCH is entered, the algorithm will attempt to obtain the cell_size from the ORIGIN command or from the last cell in the preceding region. Otherwise, cell_size should be entered.) If DISTANCE is one of the options specified, cell_size will be normalized to provide an exact integral number of cells, thus ensuring that grid lines fall precisely on the ends of the region_size.

There is a constraint on these parameters. The equation

$$d < I^2 \delta x < (2I - 1)\, d$$

must be satisfied for the grid to be monotonic. Only choosing extremely non-uniform parameters can violate this constraint. An error message will result.

Restrictions:

1. There is an upper limit to the number of spatial grid points in each coordinate.
2. There is a separate upper limit to the product in all coordinates, i.e., the total number of cells.
3. The grid must be monotonic: the parameter constraint described above must be satisfied.
4. No grid should be entered for the third coordinate (X3) in a 2D simulation.

See Also:

GRID ORIGIN, Ch. 11
GRID EXPLICIT, Ch. 11
GRID UNIFORM, Ch. 11
GRID PADE, Ch. 11
GRID SIN / COS, Ch. 11

Examples:

A simulation of a coaxial cable in cylindrical coordinates requires a grid between the inner and outer radii (R_inner and R_outer). We can shift the radial origin to R_inner and divide the gap into 20 quadratically varying cells, where the last cell is three times as large as the first cell, with the commands,

```
GRID  ORIGIN  X2  R_inner  ;
gap  =  R_outer  -  R_inner  ;
Ncells  =  20  ;
dR_ave  =  gap  /  Ncells  ;
dR_first  =  0.5 * dR_ave  ;    !  dR_first  +  dR_last  =  2  dR_ave.
dR_last  =  2 * dR_ave ;
GRID QUADRATIC X2 DISTANCE gap FIRST dR_first CELLS Ncells ;
```

Note that the DISTANCE, FIRST, and CELLS options are used to satisfy the requirement for three parameters.

GRID PADE Command

Function: Creates a Pade function grid in a region.

Syntax:

```
GRID PADE { X1, X2, X3 }
          [ DISTANCE region_size ]
          [ FIRST { MATCH, cell_size } ]
          [ LAST cell_size ]
          [ CELLS cells ] ;
```

Arguments:

region_size	-	length of the region in meters or radians.
cell_size	-	cell size in meters or radians.
cells	-	number of cells in the region.

Description:

The PADE option results in a Pade function-spaced grid over a region. The ith full-grid point, xif, in a particular region is given by the formula

$$xi = (a + b\,i) / (c + d\,i) \ .$$

The use of the Pade functional prescription is recommended for a region requiring rapid variation in the cell_size.

The PADE algorithm requires only three parameters to specify the grid completely. As indicated in the syntax, there are four options, and you must select three. If the parameters are either under- or over-specified, an error condition will result. The options are DISTANCE, FIRST, LAST, and CELLS. (The FIRST option refers to the first cell. If MATCH is entered, the algorithm will attempt to obtain the cell_size from the ORIGIN command or from the last cell in the preceding region. Otherwise, cell_size should be entered.)

If DISTANCE is one of the options specified, cell_size will be normalized to provide an exact integral number of cells, thus ensuring that grid lines fall precisely on the ends of the region_size. (In the Pade prescription, the cell_size refers to the cell half-grid, whereas in all other options, it refers to full-grid.)

Restrictions:

1. There is an upper limit to the number of spatial grid points in each coordinate.
2. There is a separate upper limit to the product in all coordinates, i.e., the total number of cells.
3. No grid should be entered for the third coordinate (X3) in a 2D simulation.

See Also:

GRID ORIGIN, Ch. 11
GRID EXPLICIT, Ch. 11
GRID UNIFORM, Ch. 11
GRID QUADRATIC, Ch. 11
GRID SIN / COS, Ch. 11

Examples:

A simulation of a coaxial cable in cylindrical coordinates requires a grid between the inner and outer radii (R_inner and R_outer). We can shift the radial origin to R_inner and divide the gap into 20 Pade function cells, where the last cell is three times as large as the first cell, with the commands

```
GRID  ORIGIN  X2  R_inner  ;
gap  =  R_outer  -  R_inner  ;
Ncells  =  20  ;
dR_ave  =  gap  /  Ncells  ;
dR_first  =  0.5 * dR_ave  ;    !  dR_first  +  dR_last  =  2  dR_ave.
GRID PADE X2 DISTANCE gap FIRST dR_first CELLS Ncells ;
```

Note that the DISTANCE, FIRST, and CELLS options are used to satisfy the requirement for three parameters.

12. OUTER BOUNDARIES

This Chapter covers the following commands:

SYMMETRY
PORT
RESONANT_PORT
OUTGOING (2D simulations only)
FREESPACE
MATCH (2D simulations only)
IMPORT

You can use these commands (along with conducting objects) to define the outer boundaries of the simulation. To use them effectively requires knowledge of a few simple conventions:

1. These commands may be used only after the grid has been generated.

2 In general, outer boundaries are applied on surfaces (spatial lines in 2D simulations and spatial areas in 3D simulations) which are conformal with one of the coordinate axes. However, the FREESPACE command can be applied only in a volume (area in 2D simulations), and the downstream surface of this volume is part of the outer boundary perimeter.

3 The outer boundary perimeter must completely enclose the simulation volume. The perimeter can have any shape and be made up of any combination of outer boundaries and conductors, but it must not contain any "holes." Holes can cause serious errors which may not be readily observable.

4 Some outer boundaries require a sense, or direction, as indicated by the syntax arguments, {POSITIVE, NEGATIVE}. The sense is always from outside the perimeter to inside. If this is in the positive direction (increasing coordinate), enter POSITIVE; otherwise, enter NEGATIVE.

The SYMMETRY commands provide axial, mirror, and periodic symmetry boundaries. You must apply all such boundaries explicitly, since none are assumed. Axial symmetry is required in polar coordinates if, and only if, the simulation area extends to zero radius (3D simulations only). Mirror symmetry can be used to cut simulation costs in half, and periodic symmetry is used to study a limited region of a repetitive or re-entrant device.

Other boundaries allow outgoing waves to exit the simulation and, simultaneously, incoming waves to enter. The PORT command provides a good, general-purpose boundary and is used extensively. Its main drawback is the need to specify the phase velocity precisely. It is not a broadband boundary, and it typically does not work well with multiple frequency, or broad-spectrum waves. The OUTGOING command (available only in 2D simulations) can also be used for outgoing and incoming waves. Its advantages over PORT are that phase velocities need not be specified, and that it tolerates a broader spectrum. However, it is more susceptible to instability.

The FREESPACE command only treats outgoing waves. It applies a special (non-physical) resistivity within a spatial volume in 3D simulations (or spatial area in 2D simulations). Its advantage is that it is very broadband and will handle a spectrum of phase velocities simultaneously. Its disadvantages are sensitivity to space charge and a possible need to tune the resistivity parameters.

There are also special-purpose boundaries. If external impedances (outside the perimeter) are important, you can use the MATCH command to match a transmission line to the simulation. (Transmission lines are discussed in the next Chapter.) The spatial line at which the match is made is considered part of the simulation perimeter. The IMPORT command can be used to introduce particles and fields from another simulation (EXPORT, Ch. 25).

SYMMETRY Command

Function: Applies outer boundaries for axial, mirror, and periodic symmetry.

Syntax:

SYMMETRY AXIAL { line, area } ;
SYMMETRY MIRROR { line, area } ;
SYMMETRY PERIODIC { line, area } { line, area } ;

Arguments:

line - name of a spatial line, defined in LINE command (2D simulations only).
area - name of a spatial area, defined in AREA command (3D simulations only).

Description:

The SYMMETRY commands apply symmetry conditions on conformal surfaces which thereby become part of the simulation perimeter. In 2D simulations, the surfaces are represented by lines; in 3D simulations, they are represented by areas. The symmetry boundaries can be applied in various combinations. They affect particle transformations as well as the electromagnetic fields. The options available are AXIAL, MIRROR, and PERIODIC.

The AXIAL is needed if (and only if) the coordinate system is non-cartesian and contains the region of zero radius (or $\theta = 0$ or π in spherical coordinates). (In 3D simulations, the axis of symmetry must be represented by an area which includes the full range of the azimuthal angle.) MAGIC2D will automatically generate this command when the coordinate system is CYLINDRICAL and the r-grid begins at r=0, if it has not been explicitly entered by the user. MAGIC3D will automatically generate this command when the coordinate system is CYLINDRICAL or POLAR and the r-grid begins at r=0, if it has not been explicitly entered by the user.

MIRROR, as the name suggests, replicates bilateral symmetry, which means simply that whatever occurs in the simulation space also occurs simultaneously in the virtual space opposite the boundary.

PERIODIC is used to effect a repetitive structure or to close polar coordinates when the full range of polar angle (2π) is required. Note that PERIODIC requires two distinct spatial areas (which may be spatially coincident). MAGIC2D will automatically generate this command when the coordinate system is POLAR and the range of the ϕ-grid spans 2π, if it has not been explicitly entered by the user. MAGIC3D will automatically generate this command when the coordinate system is CYLINDRICAL or POLAR and the ϕ-grid spans 2π, if it has not been explicitly entered by the user.

Restrictions:

1. The surface where the boundary condition is applied must be conformal in one coordinate.

2. The symmetry boundaries must be compatible with the models used in the simulation to retain physical fidelity. For example, the axial boundary must be used only at zero radius in polar coordinates (or $\theta = 0$ or π in spherical coordinates), the mirror and periodic boundaries should only be used on flat (never curved) surfaces, the periodic boundaries must appear on opposite sides, and so on.

3. The algorithm logic assumes conventional use of these boundaries, e.g., mirror symmetry near the edge (not the middle) of the simulation, axial symmetry at zero radius, etc. If you attempt unconventional uses, please be aware that the algorithm logic may thwart your intent.

4. Axial symmetry is available in spherical coordinates, but may not contain the origin.

5. SYMMETRY AXIAL results in a modified Courant condition, in which the first radial cell appears to be smaller than it actually is. The effective size is

$$\sqrt{2/3}\,\delta r \cong 0.82\,\delta r$$

See Also: **SYSTEM, Ch. 10**

References:

B. Goplen, R. S. Coats, and J. R. Freeman, "Three-Dimensional Simulation of the PROTO-II Convolution," Mission Research Corporation Report, MRC/WDC-R-055, presented at the 4th IEEE Pulsed Power Conference, June 6-8, 1983.

PORT Command

Function: Applies an outer boundary for outgoing (and incoming) waves.

Syntax:

```
PORT  { line, area } { POSITIVE,  NEGATIVE  }
      [ PHASE_VELOCITY   phase_velocity ]
      [ EXPANSION  scale  ]
      [ INCOMING   f(t)
            { FUNCTION {E1,E2,E3} g(x1,x2,x3) [ {E1,E2,E3} g(x1,x2,x3) ] ,
              FILE  file {E1,E2,E3} record  [ {E1,E2,E3} record ],
              LAPLACIAN  nobjects  object,voltage ...
                  [ ITERATIONS  iterations ]
                  [ OMEGA  omega ]
                  [ INITIALIZATION { X1, X2, X3, potential(x1,x2,x3) } ] }
            [ NORMALIZATION
                  { PEAK_FIELD {E1,E2,E3} point ,    VOLTAGE  line  }
                  [CIRCUIT time_constant desired(t) obs$name] ] ;
```

Arguments:

line	- name of a spatial line, defined in LINE command (2D simulations only).
area	- name of a spatial area, defined in AREA command (3D simulations only).
phase_velocity	- relative phase velocity, v/c (unitless).
scale	- geometric scaling factor (unitless, default is unity)
f(t)	- temporal function for incoming wave (V or V/m), defined in FUNCTION command.
g(x1,x2,x3)	- spatial-profile function for incoming wave (arbitrary units), defined in FUNCTION command.
file	- name of file containing spatial-profile data
record	- name of record containing data for specified field component.
nobjects	- number of independent conducting objects at the boundary.
object	- name of a spatial object, defined in AREA (2D) or VOLUME (3D) command.
voltage	- conductor voltage for Laplacian solution (V).
iterations	- number of relaxation iterations (default = 10,000).
omega	- over-relaxation constant (default = 1.0).
potential	- initial guess for potential distribution, defined in FUNCTION command.
point	- name of a spatial point, defined in POINT command.
time_constant	- circuit time constant (sec).
desired	- desired value of the observe measurement.
obs$name	- observe measurement.

Description:

The PORT command specifies a boundary which allows outgoing waves and particles to escape and incoming waves to enter. You can apply it on any conformal surface, which thereby becomes part of the simulation perimeter. The typical simulation produces outgoing waves due to scattering from geometric irregularities and plasma interactions within the simulation perimeter. You may also define arbitrary incoming waves. Thus, incoming waves, outgoing waves, and charged particles may be incident upon the boundary.

First, specify where the boundary is applied, using a line in 2D simulations and an area in 3D simulations. The surface must be conformal with one coordinate and part of the simulation perimeter. You must also enter the sense of the surface, which is always from outside the perimeter to inside. If this is in the positive direction (increasing coordinate), enter POSITIVE. Otherwise, you must enter NEGATIVE.

The algorithm uses the specified PHASE_VELOCITY to allow outgoing fields to exit and to separate the incoming and outgoing fields. The default of unity is suitable only for TEM waves in vacuum (TEM is a special case of TM). The performance of this algorithm depends critically on matching the actual phase velocities. For example, the reflection coefficient for an outgoing wave with phase velocity, v, is

$$\rho = \left(\frac{v - v_{port}}{v + v_{port}} \right)^2$$

which gives perfect transmission only if the velocities match. If multiple phase velocities are present in a single electromagnetic mode, inaccuracy is unavoidable.

The EXPANSION option is used in special cases in which the cross-sectional area in the direction of propagation grows (or shrinks), e.g., radial propagation in a conical coax. In most applications, e.g., axial propagation in a cylindrical coax, the area does not change with distance. However, if the area does change, the fields must change also. Scale multiplies fields at the boundary (x) to produce the correct magnitude one cell inside the perimeter (x-dx). For example, for a radially outgoing boundary in polar coordinates, scale would be set to R/(R-dR), since the fields inside the boundary are compressed. This accounts for changing area but not changing impedance. Thus, it works well for radial propagation in spherical coordinates (a conical coax) but not in cylindrical or polar, where continuous scattering occurs. We recommend avoiding such applications if possible. Sometimes, a "dog-leg" section can be added so that the PORT boundary is applied in a region of constant impedance. If this is not possible, scale can sometimes be varied in an ad hoc manner to achieve acceptable results.

The INCOMING keyword provides the option to specify incoming waves, i.e., waves incident on the boundary from outside the perimeter. In this case, the total field in the boundary is the sum of the incoming and outgoing fields, which we write in the form,

$$E(x,t) = E_i(x,t) + E_o(x,t) = f(t)g(x) + E_o(x,t)$$

For the incoming wave, you must specify the temporal function, f(t), and the spatial profile function, g(x), where x is the coordinate transverse to the direction of propagation (parallel to the boundary). (Under NORMALIZATION, if you enter VOLTAGE, g(x) will be re-normalized so its integral over the boundary gives unity. Then f(t) will carry units of volts.) In general, discontinuities in f and its first derivative should be avoided. For example, to launch a sinusoidal wave, we recommend multiplying the sine wave by an envelope which goes to unity in a few cycles (see Examples, below).

In 2D simulations, you specify one transverse field component and its spatial profile; in 3D simulations, you must specify two. The spatial profile function, g, for a TEM wave may be simple, e.g., 1/r for axial propagation in a co-axial geometry. For more complex geometries or non-TEM waves, g(x) is generally more complex; e.g., Bessel functions for cylindrical waveguide modes. Therefore, the spatial shape of the wave may be entered in three different ways: by supplying a function, by reading a record from a file, or by solving the LAPLACIAN equation numerically (3D simulations only).

The FUNCTION option is most useful when the spatial shape is a well-known function. You enter one component and its shape (2D) or two components and their shapes (3D). The FILE option can be used if you have previously simulated the port geometry specifically to obtain the profile. The file from the previous simulation would be created using a DUMP command or option (DUMP, Ch. 25), and the record names would be available in the TOC file. The LAPLACIAN option (3D simulations only) computes the solution to the LAPLACIAN in the specified area, which is the correct spatial profile for a TEM wave. You must supply the number of conducting objects and a list of their names and voltages. The voltages are relative and none should exceed a value of unity. The algorithm iterates to converge to a solution. You may supply the number of iterations, an over-relaxation coefficient, omega, and a guess for the potential to start the relaxation process. The INITIALIZATION option allows you to specify either an axis for a linear gradient or a function as the initial guess.

The NORMALIZATION option allows you to re-normalize the amplitude of the incoming wave. With PEAK_FIELD, you specify a field component and a point. Both spatial profiles will be scaled equally to provide the specified field with an amplitude of unity at this point. The VOLTAGE option scales the spatial profiles to give a voltage of unity through the specified line. Thus, the temporal function, f(t), carries the units of the incoming wave (V/m or V).

The CIRCUIT option introduces a feedback loop which rescales the driving function, f. It causes the total (incoming plus outgoing) field to equal the specified incoming field. As the equation above shows, they normally differ by the outgoing component. It compares the desired value with the measurement specified by the obs$name variable. The adjustment to the rescaling function is mediated by the time_constant according to:

$$\text{rescale}(t) = \exp\left[\frac{1}{\delta t}\left(1 - \frac{\delta t}{\tau}\right)\int_0^t\left(1 - \frac{\text{obs\$name}(t')}{\text{desired}(t')}\right)dt'\right] ,$$

where δt is the time step. Most geometries have a practical lower limit on the time_constant, which is typically a round-trip transit time, or perhaps a couple periods of an oscillation. The user must determine this practical limit using a combination of physical intuition, common sense, experience, and trial-and-error. Attempting to set the time_constant below the practical limit will result in dramatic overshoot of the desired behavior, oscillation, and possibily even instability. A typical use for the CIRCUIT option would be to specify a source of a particular desired power, rather than voltage. In such a case, the obs$name measurement would come from an OBSERVE FIELD_POWER S.DA command. Frequently the voltage source is oscillatory so that the feedback adjusts the amplitude in such a manner as to arrive at some desired cycle-averaged quantity, such as power. In such a case, it is necessary to use the FILTER option in the OBSERVE command to arrive at an appropriate cycle-averaged measurement.

The PORT command can easily be used to produce the so-called "port approximation," which can be useful in linear beam and other applications. If you set the phase_velocity to zero, the outgoing wave will be forced to vanish at the boundary. Then, the total field in the boundary will be just the incoming field, or f(t) times g(x1,x2,x3). In employing this approximation, it is good practice to obtain the spatial profile from a separate simulation using the same geometry (assuming it is unknown). It may also be sensible to modulate the temporal function, f(t), as described above.

Restrictions:

1. The surface (line in 2D simulations and area in 3D simulations) where the outer boundary is applied must be conformal.

2. The electromagnetic time step must be less than the cell size (in the direction of propagation) divided by phase_velocity times c for Courant stability, $\delta t \leq \delta x / \beta c$.

3. The value of f(t) and its first derivative at t = 0 should vanish. (Before t = 0, the effective value of f(t) is zero, regardless of the function's value.)

4. The spatial profile function, g(x1,x2,x3), must be correct for the geometry and the specified incoming wave. Artificial scattering will occur at the boundary if g is incorrect.

5. The FILE and LAPLACIAN options are available only in 3D simulations.

6. The CIRCUIT option can be used to maintain a specified constant amplitude following a transient rise, but never to obtain a continuously varying amplitude.

7. Avoid close proximity to structural singularities, e.g., corners, which produce fringe fields in all direction as this may lead to instabilities.

8. Avoid close proximity to particles, which can distort phase velocities and cause artificial field reflections.

9. Avoid applications involving impedance change. The pre-eminent example is the radially outgoing wave in cylindrical coordinates. We would recommend building an elbow and letting the wave out axially. If this is not possible, then use scale, but test the results!

See Also: MAXWELL, Ch. 17
OUTGOING, Ch. 12
FUNCTION, Ch. 6

References:

B. Goplen, "Boundary Conditions for MAGIC," Mission Research Corporation Report, MRC/WDC-R-019, 23rd Annual Meeting, APS Division of Plasma Physics, 12–16 October, 1981.

B. Goplen, R. S. Coats, and J. R. Freeman "Three-Dimensional Simulation of the PROTO-II Convolution," Mission Research Corporation Report, MRC/WDC-R-055, 4th IEEE Pulsed Power Conference, June 6–8, 1983.

Examples:

In this 3D simulation, the incoming wave is a sinusoidally varying, one-volt, TEM wave, applied to the "inlet" at the left end of a coaxial cable with radii, R_cathode and R_anode. Note the modulation envelope applied to the sinusoidal function to ensure that the derivative of f(t) vanishes at t = 0.

```
LINE gap CONFORMAL R_cathode,0,Z  R_anode,0,Z ;
AREA inlet CONFORMAL R_cathode,0,Z  R_anode,TwoPi,Z ;
FUNCTION g1(x1,x2,x3) = 1 / x1 ;
FUNCTION g2(x1,x2,x3) = 0 ;
FUNCTION f(t) = SIN (omega*t ) * (1.0 - EXP( - omega*t/25 )) ;
PORT inlet POSITIVE INCOMING f
     FUNCTION E1 g1 E2 g2
     NORMALIZATION VOLTAGE gap ;
```

RESONANT_PORT Command

Function: Applies an outer boundary that models the effect of a resonant cavity or lumped circuit.

Syntax:

```
RESONANT_PORT { line, area } { POSITIVE, NEGATIVE } frequency
              IMPEDANCE voltage_line Rshunt_to_Q Q freq_offset kappa_C kappa_L
              GAP_FIELDS {gap_area, gap_volume }
                    { FUNCTION el(x1,x2,x3) e2(x1,x2,x3) e3(x1,x2,x3),
                      READ file format e1_record e2_record e3_record
                                 [ SHIFT x1_shift x2_shift x3_shift] }
              [ EXTERNAL Q_external [ INPUT power [ PHASE degrees ] ] ]
              [ RELAXATION n_cycles ]
              [ ALGORITHM { BEAM_CURRENT, GAP_CURRENT, DIAGNOSTI } ] ;
```

or

```
RESONANT_PORT { line, area } { POSITIVE, NEGATIVE } JOIN resonant_port ;
```

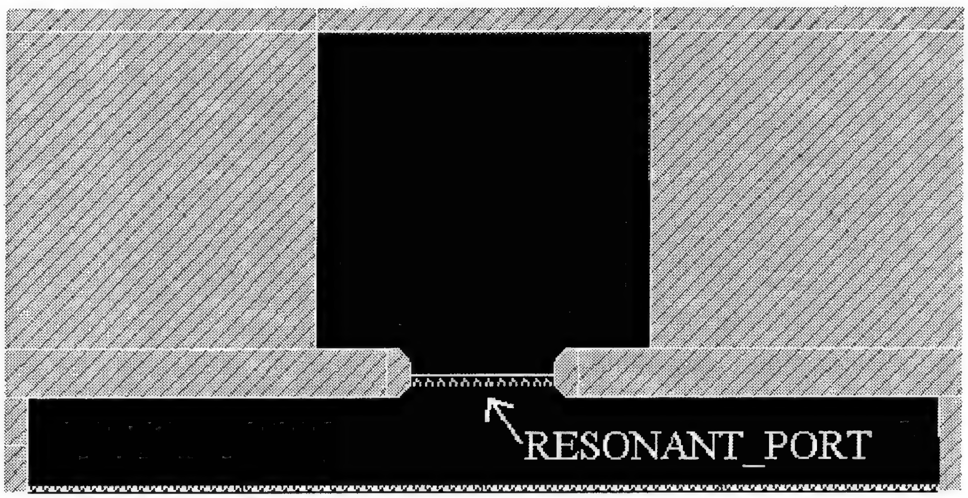
Arguments:

line	- name of a spatial line, defined in LINE command (2D only).
area	- name of spatial area defined in AREA command for (3D only).
frequency	- applied oscillation frequency in simulation, in Hz.
voltage_line	- name of voltage normalization line, defined in LINE command.
Rshunt_to_Q	- energy impedance, in ohms, consistent with the voltage_line.
Q	- loss parameter, unitless.
freq_offset	- applied frequency minus cavity resonant frequency, in Hz.
kappa_C	- fraction of capacitance treated by boundary, $0 \le$ kappa_C ≤ 1.
kappa_L	- fraction of inductance treated by boundary, $0 \le$ kappa_L ≤ 1.
gap_area	- name of the gap area, defined in AREA command (2D only).
gap_volume	- name of the gap volume, defined in VOLUME command (3D only).
e1, e2, e3	- the oscillation's electric fields, defined in FUNCTON commands.
file	- name of a file from which to read the oscillation's electric fields.
format	- format of the file (ASCII or BINARY).
e1_record, ...	- name of oscillation's electric field records in the file.
x1_shift, ...	- coordinate shift applied to read-in electric fields.
Q_external	- additional external loss parameter, unitless.
power	- input power, in watts.
degrees	- phase of input signal, in degrees.
n_cycles	- number of cycles for relaxation of the boundary field; default is 1.
resonant_port	- name of previously-defined RESONANT_PORT line/area with which to join.

Description:

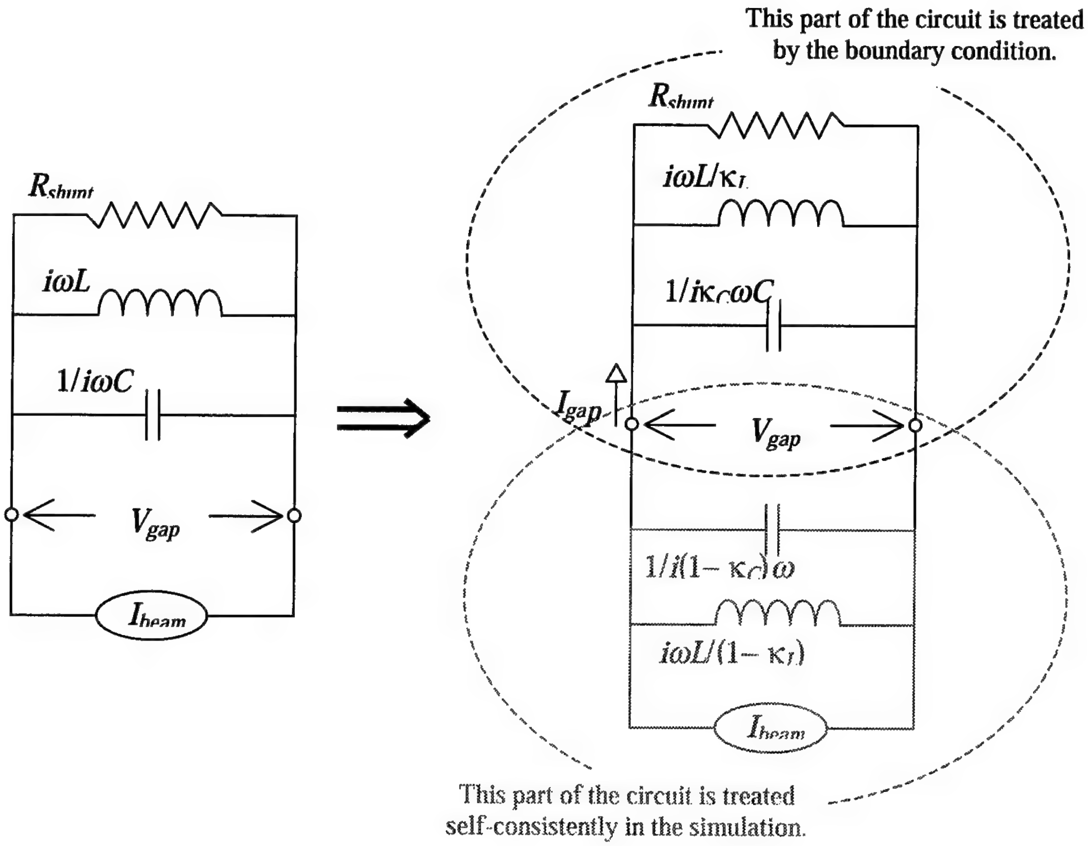
The RESONANT_PORT command provides an outer boundary condition with very special properties that mimic the behavior of a cavity or lumped-circuit, without actually having to create the geometry of the cavity or lumped-circuit in the simulation. The use of this boundary condition, in lieu of the actual geometry, obviously introduces simplification and approximation to a simulation, and the user must be alert to the possibility of an associated loss in fidelity. Keeping this in mind, careful use of the boundary condition can result in significant reduction in simulation size and run time, by reducing the total volume of the simulation, and shortening the time required to achieve cavity saturation.

Typically, the RESONANT_PORT boundary is placed in the open gap between the noses of a re-entrant cavity structure, as illustrated in the following figure. For klystron type devices, RESONANT_PORT can successfully substitute for the input and initial buncher cavities. The lumped-circuit approximation is less accurate as the beam current modulation increases, so the boundary condition should not be used as a substitute for the penultimate or output cavity. Experience with the boundary condition also indicates that failure occurs earlier when the drive and cavity frequencies are separated near the limit of the cavity bandwidth, as occurs in wide-band applications of the klystron concept. The RESONANT_PORT boundary condition continues to undergo development in order to improve its range of applicability.



The boundary mimics a very simple parallel RLC circuit. The voltage across this parallel circuit, V_{gap}, is used to apply the electric field boundary condition on the RESONANT_PORT line or area, and this voltage is arrived at from a simple impedance relationship to either an induced beam current or an effective gap current. You must specify a voltage_line that ties the circuit voltage, V_{gap}, to the geometry of the simulation. In addition, you must specify the IMPEDANCE parameters in the form of the energy impedance, $R_{shunt}/Q=(L/C)^{1/2}$, the loss parameter, Q, and the resonant frequency, $f_0=(LC)^{-1/2}/2\pi$. In a 2D simulation, the voltage_line may be the same as the RESONANT_PORT line, or it may be some other line, e.g., the cylindrical axis. Note that the stored energy of the circuit is defined as $U={}^1/_2 V_{gap}{}^2/(R_{shunt}/Q)$, so different voltage_lines will have slightly different values of R_{shunt}/Q in order to preserve the stored energy of the circuit. You must insure that your value of R_{shunt}/Q is consistent with your voltage_line.

In addition to these circuit parameters, the user must also specify the fractions of the circuit capacitance and inductance, κ_C and κ_L, which are treated by the boundary. Non-unity fractions occur, because the resonator's volume invariably encompasses part of the simulation volume adjacent to the boundary, and the field energy within this region is already treated self-consistently in the simulation and should not be treated by the boundary condition. This split is illustrated in the following figure, where the inductance and capacitance have been split into two parts, that treated by the boundary condition and that treated self-consistently by the simulation. The fractions correspond quite precisely to the fraction of electric and magnetic energy outside the surface of the RESONANT_PORT boundary in an actual cavity. A good way to determine these fractions is to OBSERVE the stored energy in an actual cavity having the same R_{shunt}/Q, Q, and f_0 as the RESONANT_PORT boundary. For a typical cavity geometry, κ_C will be in the 0.5–0.75 range, e.g., half to three-quarters of the capacitance is treated by the boundary, and κ_L will be very close to unity, e.g., nearly all the inductance is treated by the boundary. It is especially important to have accurate values for these fractions when the GAP_CURRENT algorithm is being used; by contrast, the BEAM_CURRENT algorithm is insensitive to these fractions.

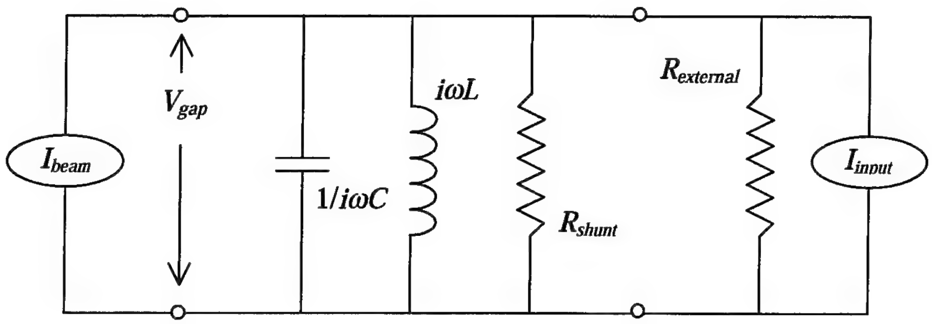


You must also specify the electric field pattern of the resonator oscillation in the gap region. These GAP_FIELDS provide the electric field profile at the boundary and are also needed to compute the induced current. The normalization of the field patterns is not important. You first specify the gap_area (2D) or gap_volume (3D) over which the fields are significantly non-zero. There are two possible ways to input the GAP_FIELDS. The first is to provide three analytic FUNCTIONs of the fields; the second is to READ three records from a file containing the field information. Any such file must be a FLD-type dump-file in either ASCII or binary format and would typically be produced from a simulation of an actual cavity, using the TABLE command to output the field patterns over the desired gap_area or gap_volume. The record names for the field patterns would be found in the TOC file from the same simulation. (The actual cavity simulation can also be used to provide the κ_C and κ_L values via OBSERVE FIELD_ENERGY commands.) The user might alternately wish to use existing field patterns from some other source, in which case, it would be necessary to convert the existing data into the FLD-type dump-file format. If the coordinate systems of the field patterns in the file and simulation do not have the same origin, the SHIFT option can be used to add a fixed translation vector to the coordinates in the file. It is essential to have complete and accurate gap fields in the entire gap region for the BEAM_CURRENT algorithm; by contrast the GAP_CURRENT algorithm is largely insensitive to the gap fields.

The internal circuit quantities of a RESONANT_PORT can be diagnosed with the OBSERVE RESONANT_PORT command.

By default, there is no additional external load or input power in the circuit model of the boundary. An EXTERNAL option allows the user to add the external circuit elements shown below, e.g., a load with loss parameter $Q_{external}=R_{external}/(R_{shunt}/Q)$, and an optional ideal current source whose input power ($=\frac{1}{2}V_{gap}I_{input}^*$) is relaxed to the user-specified value. The total Q of the circuit is then given by $1/Q_{total}=1/Q+1/Q_{external}$. In

situations involving multiple-input cavities, the relative PHASE between the input signals can also be set as an option.



The RESONANT_PORT boundary condition must perform significant relaxation operations. The default time scale for these operations is one cycle of the oscillation, but this may result in an unacceptable level of noise or startup jitter, and the user has the option of setting the RELAXATION time scale to a larger number of cycles. In addition, the RESONANT_PORT boundary contains two different ALGORITHMs for driving the circuit, e.g., the BEAM_CURRENT and the GAP_CURRENT algorithms. The requirements, advantages, and disadvantages of the two methods have been previously highlighted and are discussed in detail below. A third AGLORITHM option allows RESONANT_PORT to be placed in an actual cavity simulation as a DIAGNOSTIC, rather than as an outer boundary condition, for the express purpose of observing the two drive terms and comparing them with the actual cavity signals.

The gap impedance as derived from the above circuit diagrams is:

$$Z_{gap} = \frac{V_{gap}}{I_{gap}} = \frac{1}{i\kappa_C \omega C + \frac{\kappa_L}{i\omega L} + \frac{1}{R_{shunt}}} = Q(R_{shunt}/Q)\frac{1+i\Delta(\omega)}{1+\Delta^2(\omega)} \quad ,$$

where

$$\Delta(\omega) = Q\left(\kappa_C \frac{\omega}{\omega_0} - \kappa_L \frac{\omega_0}{\omega}\right) \quad .$$

When the BEAM_CURRENT algorithm is used, an induced beam current, I_{beam}, is computed from the particle current, $\mathbf{J}(\mathbf{r})$, following Ramo's theorem, e.g.,

$$I_{beam} \equiv \int_{gap\_region} d^3r\, \mathbf{J}(\mathbf{r}) \cdot \mathbf{E}_{mode}(\mathbf{r}) \quad ,$$

where $\mathbf{E}_{mode}(\mathbf{r})$ is the resonator oscillation fields normalized for unit voltage across volt_line. The BEAM_CURRENT algorithm drives the boundary directly from I_{beam}, which is transformed to I_{gap}, as illustrated in the circuit diagram (equivalent to setting $I_{gap}= I_{beam}$ and taking $\kappa_C=\kappa_L=1$). The BEAM_CURRENT has the advantage of driving the circuit to saturation in very few cycles, thereby saving considerable run-time. This algorithm is, however, susceptible to inconsistency because of the physical separation of the driving term, e.g., the beam and the boundary. Evidence of inconsistency usually consists of particle energy loss or gain that is significantly different from that reported from the circuit. The primary cause of inconsistency is beam space-

charge effects, which become important when the frequency offset is not zero or when the gap voltage is significant compared to the beam voltage. Inconsistency is increased when GAP_FIELDS are not representative of actual fields in the resonator.

When the GAP_CURRENT algorithm is used, the gap current, I_{gap}, is computed from the magnetic field, $\mathbf{H}(\mathbf{r})$, along the RESONANT_PORT boundary according to

$$I_{gap} \equiv \oint_{boundary} d\mathbf{A} \cdot \left[\tilde{\mathbf{E}}_{mode}(\mathbf{r}) \times \mathbf{H}(\mathbf{r})\right] \quad .$$

The energy into both the circuit and the simulation boundary are identically given by $\frac{1}{2} V_{gap} I_{input}^*$, so that there is no danger of inconsistency. An additional advantage of the GAP_CURRENT algorithm is that it can be driven from coupling to nearby cavities, or any other source of RF, while the BEAM_CURRENT is driven solely by particle current within the area/volume of the GAP_FIELDS. The primary disadvantage of the GAP_CURRENT method is the requirement for accurate estimates of K_C and K_L, and the fact that resonator saturation is not speeded up.

There may be situations in which a cavity gap cannot be bridged by a single line or area, especially in 3D cartesian simulations. It is possible to join together multiple RESONANT_PORT boundaries and drive them all from the same circuit and GAP_FIELDS. The procedure for doing this is to create the first RESONANT_PORT boundary in the usual fashion. For the remainder of the multiple boundaries, the alternate RESONANT_PORT syntax involving the JOIN keyword should be used.

See Also: **TABLE, Ch. 21**
OBSERVE, Ch. 22

Restrictions:

1. There can be at most 20 RESONANT_PORT commands in a simulaltion.

2. The GAP_CURRENT algorithm is unstable for inaccurate values of the κ_C and κ_L user inputs.

Examples:

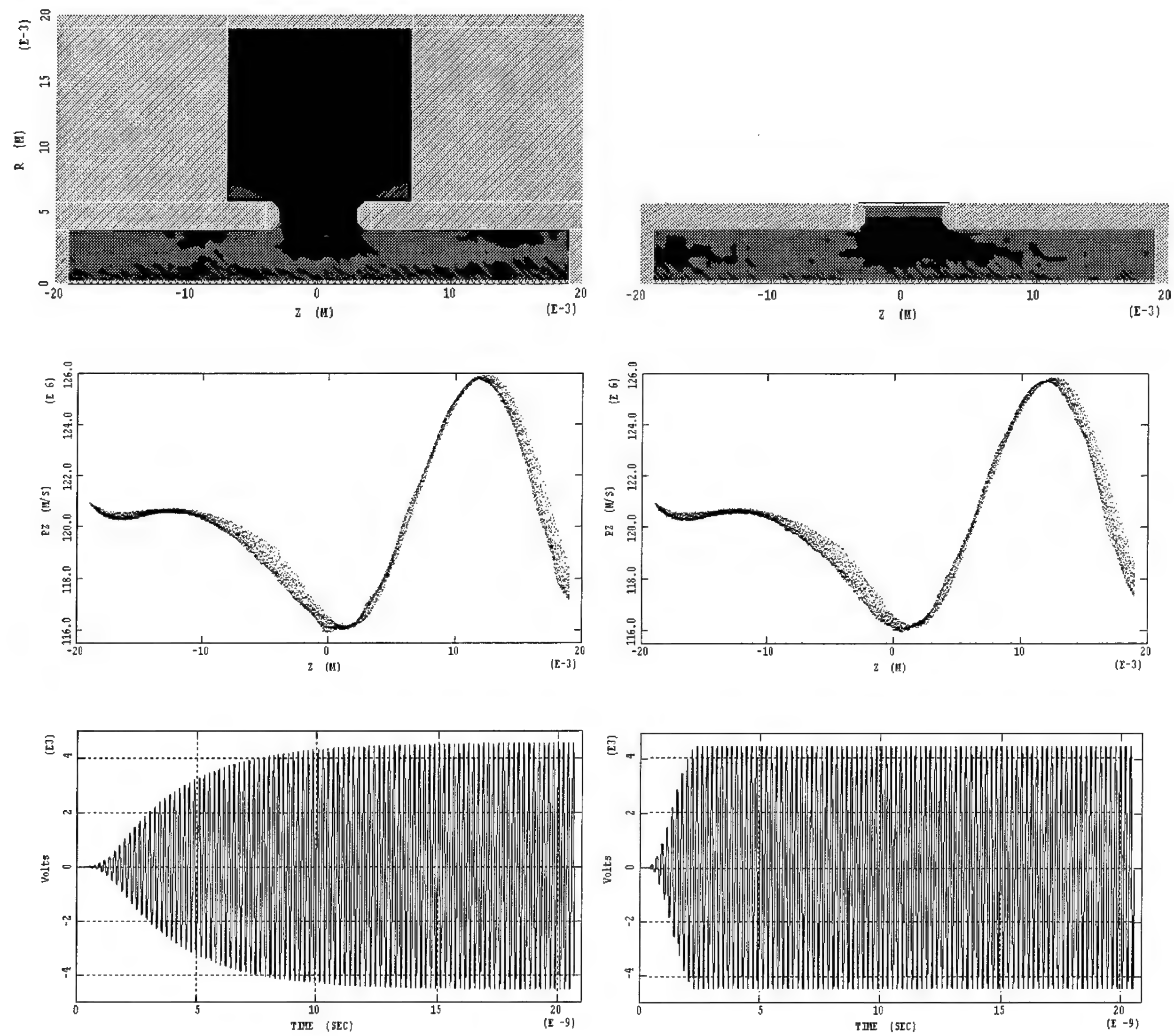
In the following example, a beam with density modulation excites a buncher cavity, which in turn imparts velocity modulation to the beam. The simulation is then rerun with the cavity removed and replaced with a RESONANT_PORT boundary. The following lines show the commands necessary to create the RESONANT_PORT boundary condition.

```
RF_DRIVE_FREQ   = 4.8272 GHZ ;
PORT_FREQ       = 4.8272 GHZ ; ! ... get this value from cold test of actual cavity
PORT_Q          = 47.6 ;       ! ... get this value from cold test of actual cavity
PORT_RSHUNTAQ = 96.50 OHMS ; ! ... get this value from cold test of actual cavity
PORT_KAPPAC     = 0.70 ;       ! ... get this value from cold test of actual cavity
PORT_KAPPAL     = 0.99 ;       ! ... get this value from cold test of actual cavity

PORT_DFREQ = RF_DRIVE_FREQ - PORT_FREQ ;
LINE GAP CONFORMAL ZBGN_GAP,R_PORT ZEND_GAP,R_PORT ;
AREA GAP_REGION CONFORMAL ZBGN_FLD,0. ZEND_FLD,RBGN_CAV ;

RESONANT_PORT GAP NEGATIVE RF_DRIVE_FREQ
              IMPEDANCE GAP PORT_RSHUNTAQ PORT_Q PORT_DFREQ PORT_KAPPAC PORT_KAPPAL
              GAP_FIELDS GAP_REGION READ EIGENMODE.FLD ASCII EZ ERHO EPHI ;

OBSERVE RESONANT_PORT GAP V_GAP ;
OBSERVE RESONANT_PORT GAP I_BEAM ;
```

```
OBSERVE RESONANT_PORT GAP I_GAP ;
OBSERVE RESONANT_PORT GAP R_CAVITY_POWER ;
OBSERVE RESONANT_PORT GAP GAP_POWER ;
```

The figures below compare snapshots of magnetic field contours and electron phase space and the time history of the gap voltage. The gap voltage is identical for the actual cavity and for the RESONANT_PORT; however, note how the gap voltage saturates much more quickly for the RESONANT_PORT than for the actual cavity.

Simulation with Actual Cavity Simulation with RESONANT_PORT



OUTGOING Command

Function: Applies an outer boundary for outgoing (and incoming) waves in a 2D simulation.

Syntax:

```
OUTGOING line { POSITIVE, NEGATIVE }
        { TE, TM, ALL }
        [ INCOMING f(t) g(x) ]
        [ ORDER norder ]
        [ COEFFICIENT alpha_1,beta_1, ... ] ;
```

Arguments:

line	- name of spatial line, defined in LINE command.
f(t)	- temporal function for incoming wave (V or V/m), defined in FUNCTION command.
g(x)	- spatial-profile function for incoming wave (arbitrary units), defined in FUNCTION command.
order	- approximation order (default = 3).
alpha_n	- nth alpha coefficient.
beta_n	- nth beta coefficient.

Description:

The OUTGOING command specifies an outlet boundary condition which absorbs any waves which are propagating outward from the simulation through the boundary. It is a generalization of the PORT command and operates in a similar manner, except that OUTGOING boundaries can absorb waves of unknown phase velocity, and indeed, different phase velocities simultaneously.

First, enter line name to specify where the boundary is applied. The line must be conformal with one coordinate and part of the simulation perimeter. You must also enter the sense of the line, which is always from outside the perimeter to inside. If this is in the positive direction (increasing coordinate), enter POSITIVE; otherwise, you must enter NEGATIVE.

It is anticipated that the OUTGOING boundary will eventually have a robust capability to launch an incoming wave with the INCOMING option. A capability is currently included which may be satisfactory for some applications. However, users are cautioned to use this option with skepticism, since it is not fully debugged. Note that if the INCOMING option is used, then either TM or TE (but not ALL) must be specified for the mode. It is possible to have two OUTGOING boundaries, one TE and one TM, at the same line_name.

Particles may exit through an OUTGOING boundary. The physical validity of this process is still under investigation; however, it is believed that the OUTGOING boundary provides a satisfactory treatment in this situation.

This boundary condition is an exceptional absorber at near speed-of-light phase velocities (normal incidence). With the ORDER option, the order controls the effectiveness of the boundary at near grazing incidence, with a higher order, resulting in better absorption at shallower angles of incidence. For example, the default order (3) results in 2% field amplitude reflection at 20 degrees incidence, 6% at 10 degrees, and 10% at 5 degrees, while at order 5, the reflection is only 0.3% at 20 degrees, 2% at 10 degrees, and 5% at 5 degrees. In general, the default order should be quite satisfactory for most applications; however, if a simulation is known to possess a significant amount of its RF power in shallow incidence waves, then a higher order can be used. With machine single precision (32 bits), there is no benefit in using an order greater than about 9.

The COEFFICIENT option allows the user to specify non-standard coefficients for the boundary, and it is generally reserved for study of the properties of the boundary itself. Under normal circumstances, this option should not be used.

Restrictions:

1. The OUTGOING command is available only in 2D simulations.

2. The spatial line where the boundary condition is applied must be conformal with one of the spatial coordinates.

3. The maximum number of OUTGOING boundaries is 25.

4. The INCOMING option cannot be used with the mode set to ALL.

5. The OUTGOING boundary may touch other spatial objects (outer boundaries or material properties) only at its endpoints. It cannot cross or overlap another spatial object which has material properties (e.g., CONDUCTOR, POLARIZER, etc.). In such applications, two separate OUTGOING boundaries must be applied.

See Also: **MAXWELL, Ch. 17**
PORT, Ch. 12
FUNCTION, Ch. 6

References:

"Free-Space Boundary Conditions for the Time Dependent Wave Equations," E. L. Lindman, *Journal of Computational Physics*, 18 (1975), pg. 66.

"Higher Order Absorbing Boundary Conditions for the Finite-Difference Time-Domain Method," P. A. Tirkas, C. A. Balanis, and R. A. Renaut, *IEEE Transactions on Antennas and Propagation*, 40 (1992), pg. 1215.

Examples:

This is an example of an OUTGOING boundary used to absorb the wave propagating along a helix TWT. Note that two separate spatial applications (above and below the helix) are required. (See Restrictions, above.)

```
SYSTEM CYLINDRICAL ;
LINE axis CONFORMAL 0,0  L,0  ;
SYMMETRY AXIAL axis ;
LINE helix CONFORMAL 0,Rh L,Rh ;
POLARIZER HELIX 45. helix ;
AREA anode CONFORMAL 0,Ra L, +Ra+dR  ;
CONDUCTOR anode ;
LINE below CONFORMAL L,0    L,Rh ;
LINE above CONFORMAL L,Rh L,Ra ;
OUTGOING below NEGATIVE ALL ;
OUTGOING above NEGATIVE ALL ;
```

FREESPACE Command

Function: Applies an outer boundary for outgoing waves.

Syntax:

```
FREESPACE { area, volume } { POSITIVE, NEGATIVE } { X1, X2,X3 }
          { TRANSVERSE, ALL, E1, E2, ..., B3 }
          [ CONDUCTIVITY f(x) ] ;
```

Arguments:

area	- name of area, defined in AREA command (2D simulations only).
volume	- name of volume, defined in VOLUME command (3D simulations only).
f(x)	- name of conductivity function (mhos/m), defined in a FUNCTION command.

Description:

The FREESPACE command allows outgoing waves to escape from the simulation. It is similar in purpose to the PORT command. However, whereas PORT is applied along a conformal surface, FREESPACE is applied over a finite region (area in 2D simulations and volume in 3D simulations). The downstream conformal surface of this region is part of the simulation perimeter.

First, specify where the boundary is applied, using an area in 2D simulations and a volume in 3D simulations. You must also enter the sense, which is always from outside the perimeter to inside, along a specified axis. If this is in the positive direction (increasing coordinate), enter POSITIVE; otherwise, enter NEGATIVE. The axis (X1, X2, or X3) specifies the axis along which propagation occurs.

The FREESPACE algorithm applies a spatially-varying conductivity to both electric and magnetic components in the field equations. This reduces the phase shift between components as both are degraded, minimizing reflection. You must specify the field components to which conductivity will be applied. The normal application involves only the transverse (propagating) field components (both electric and magnetic fields). However, in special applications, you may also enter ALL or even individual field components (E1, E2, ..., B3). In this case, a separate FREESPACE command must be given for each desired component, and care must be exercised to avoid duplication (e.g., issuing commands for ALL and E1 fields). One reason to allow multiple FREESPACE commands on the same spatial region is to be able to use different conductivity functions on different field components. For example, a special treatment of the longitudinal electric field may be used to relax space-charge fields if particles penetrate the boundary area.

The default conductivity function increases quadratically with distance in the direction of the outgoing wave, or

$$f(x_n) = \sigma_m x_n^2,$$

where the spatial coordinate, x_n, is normalized to a total length of unity over distance in the direction of propagation. The proper "zero" location, orientation, and spatial scaling are accounted for automatically by the code, so that the minimum value of conductivity (zero) will be applied at the upstream edge of the FREESPACE region, and the maximum conductivity (σ_m) will be applied at the downstream edge, which is part of the simulation perimeter. (Fields actually in the perimeter will vanish.)

The default value for the maximum conductivity, σ_m, is determined from the equation,

$$\sigma_m = \frac{4\pi c \varepsilon_0}{\lambda} = 4\pi \varepsilon_0 f = 2\varepsilon_0 \omega,$$

where λ is estimated from the distance. (It is assumed that the model will be employed over a spatial distance equal to about half the wavelength.)

The FREESPACE algorithm has a very broad bandwidth, and results are usually insensitive to the detail of the conductivity function. However, you can override the default function and enter your own function by using the CONDUCTIVITY option.

Restrictions:

1. The maximum number of FREESPACE commands is twelve.

2. The δx in the direction of propagation should be small compared to the wavelength, $\delta x < \lambda/12$, and more ideally, $\delta x < \lambda/16$. Here λ is the shortest wavelength with a significant amplitude in the simulation interior.

3. The boundary should have at least ten cells in the spatial grid in the direction of propagation, but not more than 40.

See Also: **FUNCTION, Ch. 6**
PORT, Ch. 12

Examples:

We wish to absorb the outgoing wave at the outlet end of a coaxial cable which is subjected to an incident voltage pulse at the inlet. For the sake of illustration, the default (quadratic) conductivity will be replaced by a linear conductivity having a maximum value of 10^{-3} mhos/m. This is illustrated in the following commands:

```
FUNCTION SIGMA(X) = 1.0E-3 * X ;
FREESPACE Volume_Free NEGATIVE X1 TRANSVERSE CONDUCTIVITY SIGMA ;
```

In this example, the freespace boundary is applied everywhere within Volume_Free. Note that the normalized spatial variable in the conductivity function will vary from zero to one over the axial length of this volume, irrespective of its actual physical length.

MATCH Command

Function: Matches an outer boundary in a 2D simulation to a transmission line.

Syntax: MATCH line { POSITIVE, NEGATIVE } point
transmission_line point { POSITIVE, NEGATIVE } ;

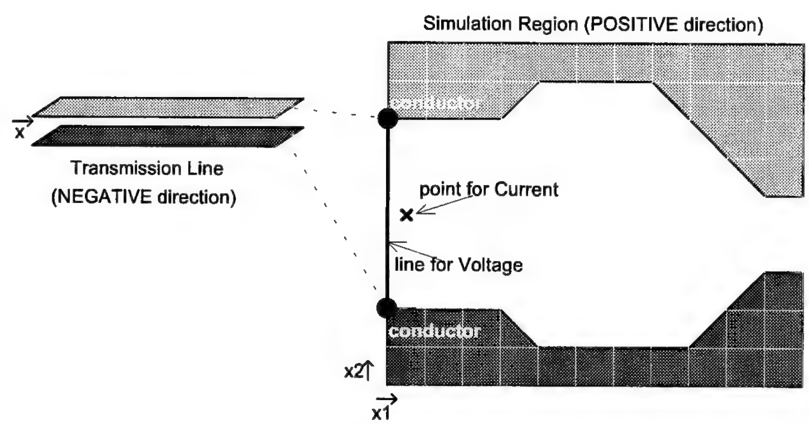
Arguments:

line - name of spatial line for the voltage match, defined in LINE command.
point - NULL, or the name of a spatial point for the current match, defined in POINT command. Also, transmission line point for voltage match.
transmission_line - name of a transmission line, defined in TRAMLINE command.

Description:

The MATCH command connects a transmission line to the simulation. Specifically, the transmission line current and voltage are coupled with the TEM fields of the simulation.

The spatial convention for the linear match is illustrated in the figure that follows. First, enter a spatial line describing the integral path for voltage. You must also enter the direction from outside the perimeter to inside. If this is in the positive direction (increasing coordinate), enter POSITIVE; otherwise, enter NEGATIVE. The point_name determines the type of match between the transmission line current and the magnetic field. If you enter NULL, an average of the magnetic field will automatically be performed over a path adjacent to the voltage integral. Alternatively, you may use the magnetic field at a single point by entering a point_name. You must also identify the transmission line and specify its end point and its direction from outside to inside, as either POSITIVE or NEGATIVE. The cell size of the transmission line and the perpendicular cell size of the simulation should be the same at the match.



Matching to a transmission line.

Restrictions:

1. All transmission lines must be specified using TRAMLINE commands.
2. Primary and secondary cell sizes must be identical at the match point (in the direction of propagation).

3. The maximum number of MATCH commands is ten.

4. This command is available only in 2D simulations.

See Also: **TRAMLINE, Ch. 13**
JOIN, Ch. 13
GRID, Ch. 11

Examples:

Consider two coaxial, six-vane magnetrons coupled by a transmission line which runs from a vane slot on the first magnetron to a vane slot of the second. The following commands are used to define the transmission line to match it to the simulation. The transmission line, CONNECT, is defined using the impedance option. The impedance is set to 5.4 ohms, and the relative dielectric constant is set to unity. The line is 1 meter in length. One MATCH command connects the transmission line to the first magnetron, and the other MATCH command connects the transmission line to the second magnetron.

```
TRAMLINE CONNECT IMPEDANCE 5.4 1.0
     FUNCTION 101 1 0.0  100 1.068E-3 10.68E-2 ;
MATCH Line_ab NEGATIVE Point_c  CONNECT 0.0  POSITIVE ;
MATCH Line_ef NEGATIVE Point_g  CONNECT 1.0 NEGATIVE ;
```

It is possible to simulate two complete (2π) magnetrons simultaneously using an extended azimuthal grid (0-4π) and two sets of periodic symmetry boundary conditions. The resulting magnetrons can be completely independent, or they can be coupled as indicated above.

IMPORT Command

Function: Applies an outer boundary for importing particles and fields.

Syntax:

```
IMPORT { line, area } { POSITIVE, NEGATIVE }
       { FILE file_prefix file_format, LAMINAR_BEAM current energy radius guide_field }
       [ SCALE scale(t) ]
       [ { START_TIME start_time, TIMER timer } ]
       [ REWIND_AT_EOF cycles, TERMINATE_AT_EOF ] ;
```

Arguments:

line	- name of spatial line, defined in LINE command (2D simulations only).
area	- name of spatial area, defined in AREA command (3D simulations only).
file_prefix	- prefix of file name.
file_format	- file format (ASCII or BINARY).
current	- beam current in amps.
energy	- beam energy in electron volts.
radius	- beam radius in m.
guide_field	- beam magnetic guide field in tesla.
scale	- field and charge scaling factor, real or defined in FUNCTION command.
start_time	- simulation time at which import from file will begin (sec).
timer	- name of timer, defined in TIMER command.
cycles	- number of times to rewind file (integer).

Description:

The IMPORT command defines an outer boundary for introducing incoming fields and particles. These may be particle and field records that were produced by an EXPORT command in a previous simulation, i.e., output from EXPORT provides input for IMPORT. Alternatively, you may specify the creation and import of an ideal laminar flow beam.

The EXPORT/IMPORT technique can be effectively used when it is desirable (and reasonable) to break a simulation into two or more parts, which can be performed more efficiently. We will use the klystron to illustrate. This technique is desirable in klystrons, because we would like to design cavity N+1 separately, without having to include all of the preceding N cavities in the simulation. It is reasonable, because the waveguides connecting klystron cavities are typically cut off, thus minimizing the backward wave, and because particles in the waveguides all move downstream. Therefore, we would EXPORT a single RF cycle at the outlet of cavity N and IMPORT this cycle over and over again at the inlet of cavity N+1. The location of the surface would be near the midpoint of the drift tube joining the cavities. (Note that EXPORT is <u>not</u> an outer boundary; its presence has no effect on the simulation. The actual outer boundary in the EXPORT simulation might be slightly downstream from the EXPORT surface. By contrast, IMPORT <u>is</u> an outer boundary and therefore part of the simulation perimeter.)

What EXPORT does is to write a time record of fields and particles to file(s). You control only the details of when during the simulation the information is output (e.g., for one full RF cycle after cavity N saturates), but the file contents are determined by the MAGIC. In general, it will contain two transverse (in the surface plane) electric field components, but in 2D TM or TE simulations, there will be only one. Only particles crossing the surface in the specified (downstream) direction will be recorded; counter-flowing particles are omitted from the file. The particle coordinates recorded will be relative to the surface, not the coordinate system. Depending on the Lorentz step_multiple, particle records may be written less frequently than field records.

What IMPORT does is to read the field and particle data files and apply the fields and particles as an outer boundary. You control the details of when during the simulation the import data is introduced (e.g., repeating one full RF cycle over and over again). If possible, it is good practice to make the spatial grid near the surface and the electromagnetic time_step, and the Lorentz step_multiple the same in the EXPORT and IMPORT simulations. If they are not the same, the IMPORT simulation will redefine the Lorentz time step to match EXPORT and then reset the electromagnetic time_step to provide an integer Lorentz step_multiple (see Restrictions, below). Thus, the particles will always be the same, but the fields may be interpolated in time as well as transverse space.

Arguments for the IMPORT command are described as follows.

The IMPORT surface must be conformal with one of the axes and part of the simulation perimeter. In 2D simulations, the surface is represented by a line; in 3D simulations, it is represented by an area, which must be conformal with the X3 coordinate. The sense of the surface is always from outside the perimeter to inside. (This is also the direction of the incoming particles and wave.) If this is in the direction of increasing coordinate, enter POSITIVE; otherwise, enter NEGATIVE. There are two methods of importing field and particle data. These are selected using the FILE or LAMINAR_BEAM keywords and are described the following paragraphs.

The LAMINAR_BEAM keyword is used to introduce a well-behaved electron beam. It is available only in MAGIC2D. The argument, current, is the total current in amps of the beam. The argument, energy, is the beam energy in volts. The argument, radius, is the beam radius at the import plane. The argument, guide_field is the desired magnetic confinement field in tesla. These arguments are used to create a smooth beam of particles. The particles are given a rotation based on the supplied guide_field strength and the radial electric field of the beam space charge. If you select a value of 0 for the guide field, the particle rotation will be created based on 1.1 times the Brillouin field as calculated for the specified beam parameters. The value used will add a system variable 'sys$bguide' which you may use to set the external magnetic field. You must supply the confining magnetic field using the PRESET command. It is your responsibility to ensure that the guide field strength used in the IMPORT command matches that in the PRESET command. See the example below.

The FILE keyword is used to introduce particle and field data created using the EXPORT command from a previous simulation. The file_prefix uniquely identifies field and particle files, i.e., 'file_prefixFLD' and 'file_prefixPAR'. The default file_prefix is the name of the input file. The file_format is either ASCII or BINARY.

The SCALE option can be used to apply a temporal scale (multiplicative factor) to the imported fields and particle charge. This can be used to introduce a full-scale result more gently into the empty simulation. (In the klystron example above, abruptly introducing the full RF cycle input would shock the cavity, creating high-frequency noise.) The default for scale is unity. The scaling function is internally trapped so that it has a minimum value of 0.0 and a maximum value of unity.

The START_TIME option can be used to specify the simulation time at which import actually begins. A system timer (TSYS$IMPORT) is automatically generated in response to this command option. The time step used for importing is also part of this system timer. You can over-ride this system timer using the TIMER option and your own timer. Note: The import process makes use of the internal time stamps in the EXPORT records to adjust the simulation time step to exactly match the time step between the EXPORT records. The process ignores the absolute time included in the record.

The keywords, REWIND_AT_EOF and TERMINATE_AT_-EOF, control the action taken when an end-of-file is encountered. REWIND_AT_EOF is used to repeat the file cycle times. ERMINATE_AT_EOF is used to terminate the simulation when the import data has been exhausted.

Restrictions:

1. Only one IMPORT command is allowed in a simulation.

2. IMPORT must follow the specification of the DURATION command.

3. IMPORT is an outer boundary and part of the simulation perimeter.

4. The coordinate systems in the EXPORT and IMPORT simulations must be the same.

5. The IMPORT surface geometry should be consistent with the EXPORT surface geometry. The spatial grids do not have to match, although it is a good idea.

6. You are responsible for making sure that the axial locations of the IMPORT and EXPORT surfaces are the same. (EXPORT data is recorded relative to the surface, not the simulation coordinates.)

7. You must not superimpose on an IMPORT boundary any other boundary, such as PORT, MIRROR, FREESPACE, or OUTGOING.

8. The Lorentz (particle kinematics) time step must be the same in the EXPORT and IMPORT simulations. (The electromagnetic time steps need not match. They are subject, however, to the usual integer step_multiple restriction relating Maxwell and Lorentz time steps.)

9. For physical validity, the EXPORT/IMPORT technique works best if there is no backward wave present. (It is typically employed in cut-off geometries, etc.)

10. The LAMINAR_BEAM option is available only in MAGIC2D.

See Also: **DURATION, Ch. 11**
MAXWELL, Ch. 17
LORENTZ, Ch. 18
DUMP, Ch. 25
EXPORT, Ch. 25

Examples:

1. In this 2D simulation, an IMPORT command is used to read both TM and TE components of the electric field as well as particles moving in the positive z-direction from an ASCII file named EXP'icase' and apply them at a line named "inlet."

```
IMPORT inlet POSITIVE  EXP'icase' ASCII ;
```

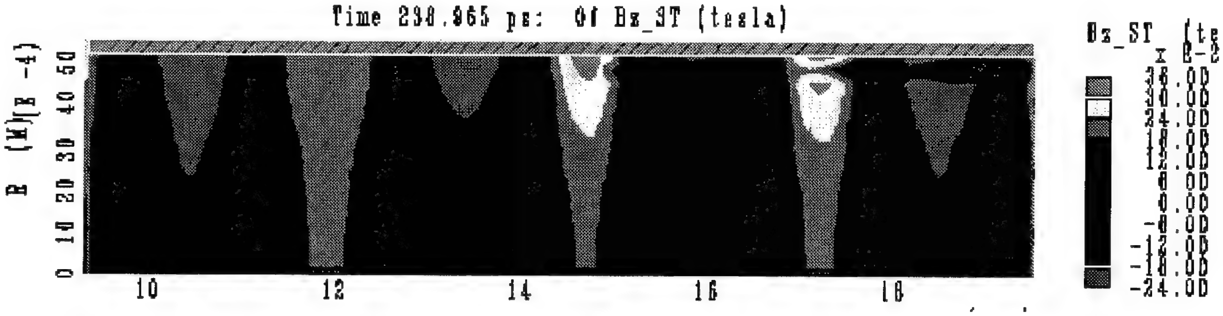
2. In this cylindrically symmetric 2D simulation, an IMPORT command is used to introduce a mildly relativistic beam (400 kV) in a drift tube of radius .2125 inches. The beam current is 275 amps. The beam fills 60% of the drift tube, and the magnetic guiding field is 0.16 tesla. The beam is introduced in the positive direction at the import plane boundary denoted by IMPORTAT. The confining magnetic field is set using the PRESET command to fix Baxial.

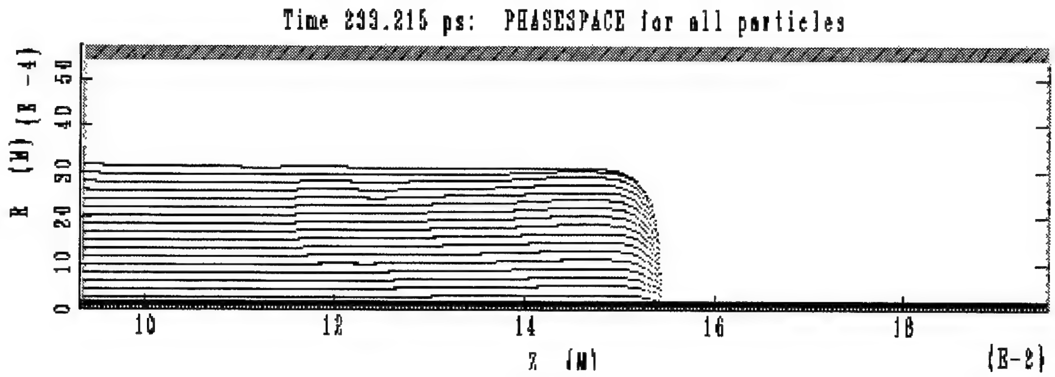
```
cyl_radius = 0.2125 inches ;
radius     = 0.60 * cyl_radius ;
```

```
current    = 275 amps ;
energy     = 400 kilovolts ;
guidefield = .16 tesla ;
IMPORT IMPORTAT POSITIVE LAMINAR_BEAM current,energy,radius,guidefield;
FUNCTION B_axial(z,r) = guidefield ;
PRESET B1ST FUNCTION B_axial ;
```

3. In this cylindrically symmetric 2D simulation, an IMPORT command is used to introduce a mildly relativistic beam (490 kV) in a drift tube of radius .2125 inches. The beam current is 275 amps. The beam fills 60% of the drift tube, and a magnetic guide field strength of 0.14 tesla is used for the import command to create the proper beam rotation. The beam is introduced in the positive direction at the import plane boundary denoted by IMPORTAT. The confining magnetic field is set using the PRESET command. Note that the first time this is run, a PPM field from PANDIRA was used to set the confining field. The field is shifted in preset to properly align with the magnetic field used to set the rotation. PRESET with PANDIRA creates the two files PANDRA01.fld and PARDRA02.fld. In subsequent simulations these may be used in place of the PRESET .. PANDIRA file.

```
Vb=490kV ;
Ib=275A;
B0=0.14Tesla  ;
rdrift=.2125inch;
ff=0.6;
remit=rdrift*ff ;
IF (pandira_new.eq.true) then ;
      PRESET B1ST PANDIRA OUTSF7 SHIFT -.541 0.;
      PRESET B2ST PANDIRA OUTSF7 SHIFT -.541 0. ;
ENDIF ;
IMPORT IMPORTLINE POSITIVE LAMINAR_BEAM Ib, Vb ,remit B0;
IF (pandira_new.ne.true) then ;
      PRESET B1ST READ PANDRA01.fld SURFACE B1st ascii SHIFT -.541 0.;
      PRESET B2ST READ PANDRA02.fld SURFACE B2st ascii SHIFT -.541 0. ;
ENDIF
```





These two figures show the contours of the axial magnetic field for a PPM stack and the particle phase space for the electron beam immersed in the PPM field.

This page is intentionally left blank.

13. TRANSMISSION LINES

This Chapter covers the following commands:

TRAMLINE (2D simulations only)
JOIN (2D simulations only)
LOOKBACK (2D simulations only)
VOLTAGE (2D simulations only)

You can use these commands to create one-dimensional transmission lines, to join them together, to allow outgoing waves to escape, and to allow incoming waves to enter. These commands are presently available only in 2D simulations.

The TRAMLINE command is used to create a transmission line. There are various options available to specify the properties of the line, which may be functions of distance; but variations in time are not allowed, nor are plasma processes. The spatial grid options duplicate the multi-dimensional options (GRID, Ch. 11).

You can use the JOIN command to connect different transmission lines together to form linear, series, and parallel circuits. The LOOKBACK command is used to allow outgoing waves to escape at a specified boundary of a transmission line. The VOLTAGE command is used to allow a specified incoming wave to enter (and outgoing waves to escape) through a specified boundary.

Although transmission lines can be modeled separately, their most important use is to extend the multi-dimensional simulation into a plasma-free region of space. (The MATCH command, which connects them, is described in Chapter 12.) The solution algorithm and time step for the transmission line will automatically match those of the multi-dimensional simulation.

TRAMLINE Command

Function: Specifies transmission-line parameters in 2D simulations.

Syntax:

```
TRAMLINE  transmission_line

  { RADII  r_outer(x)  r_inner(x)  [resistance(x)] ,
        IMPEDANCE  impedance(x)  permittivity(x) [resistance(x)] ,
        CAPACITANCE capacitance(x)  inductance(x) [resistance(x)]  }

  { EXPLICIT  CELLS  ncells
        [ SIZE  cell_size, ... ]
        [ GRID  full_grid, ...  ] ,
  UNIFORM
        [ DISTANCE  region_size ]
        [ FIRST  { MATCH,  cell_size } ]
        [ CELLS  ncells ]  ,
  QUADRATIC
        [ DISTANCE  region_size ]
        [ FIRST  { MATCH,  cell_size } ]
        [ LAST  cell_size ]
        [ CELLS  cells ] }

  [ INITIALIZE  voltage(x)  current(x) ] ;
```

Arguments:

transmission_line	- transmission line (alpha).
r_outer	- outer radius of vacuum coax (m), constant or defined in FUNCTION command.
r_inner	- inner radius of vacuum coax (m), constant or defined in FUNCTION command.
resistance	- resistance per unit length (ohms/m), constant or defined in FUNCTION command.
impedance	- impedance for arbitrary geometry (ohms), constant or defined in FUNCTION command.
permittivity	- relative dielectric constant (unitless), constant or defined in FUNCTION command.
capacitance	- capacitance per unit length (f/m), constant or defined in FUNCTION command.
inductance	- inductance per unit length (h/m), constant or defined in FUNCTION command.
cells	- number of cells in the region.
cell_size	- cell size in meters.
region_size	- length of the region in meters.
full_grid	- grid values in meters or radians.
voltage	- initial voltage distribution in volts, constant or defined in FUNCTION command.
current	- initial current distribution in Amperes, constant or defined in FUNCTION command.

Description:

The TRAMLINE command specifies parameters for transmission line models. The assumption is that regions so represented are devoid of plasma, so that impedance properties may be specified precisely. It is possible to match transmission lines to the multi-dimensional simulation. In addition, it is possible to join transmission lines to form linear, parallel, and series circuits. Boundary conditions may be applied to transmission lines using the LOOKBACK and VOLTAGE commands.

The TRAMLINE command specifies parameters defining the transmission line length, spatial resolution, and basic physical properties. The transmission_line is an arbitrary name which is used to reference the transmission line in other commands.

The first set of options specifies the impedance. Choose one of these. Any required functions are functions of position along the transmission line measured in meters. For the RADII option, the arguments are r_outer and r_inner of a vacuum coaxial cable. For the IMPEDANCE option, the arguments are impedance and permittivity. For the CAPACITANCE option, the arguments are capacitance and inductance (per unit length). If resistance is required, it can also be entered under any of the options.

The second set of options is used to specify the spatial grid. Choose one of these. The algorithms are identical with those for the multi-dimensional simulation (Part 2: GRID) and will not be discussed here. The entire grid must be specified using only one option. It is not possible to append grid onto a transmission line. All lines have a spatial origin of zero.

The final option provides the capability for non-zero initialization of the transmission line. The arguments are voltage and current functions.

The transmission line model calculates several variables of interest. These variables can be recorded by invoking the OBSERVE and RANGE commands. The following table lists the keywords and physical definitions associated with each variable.

Keyword	Definition	Notes
CURRENT	current	
POWER	power	
VOLTAGE	voltage	
CAPACITANCE	capacitance/length	1
INDUCTANCE	inductance/length	1
IMPEDANCE	impedance	1
ENERGY-CAP	capacitive energy	2,3
ENERGY-IND	inductive energy	2,3
ENERGY-TOT	total energy	2,4

1. Only accessible with the RANGE command.
2. For OBSERVE, requires a finite range, not a point.
3. For RANGE, returns energy/length.
4. Not accessible with the RANGE command.

Restrictions:

1. The maximum number of transmission lines that can be defined is ten.
2. The maximum number of total grid points that can be defined is 1000.
3. This command is available only in 2D simulations.

See Also:

SYSTEM, Ch. 10
GRID, Ch. 11
MATCH, Ch. 12
LOOKBACK, Ch. 13
VOLTAGE, Ch. 13
MAXWELL, Ch. 17
OBSERVE, Ch. 22
RANGE, Ch. 23

Examples:

Consider a coaxial, six-vane magnetron with a transmission line connected to the bottom of one vane slot. The impedance of the transmission line is matched to the characteristic impedance at the bottom of the vane slot. The end of transmission line LOAD is terminated with a VOLTAGE command. In this example, no reflection from the end of the transmission line is desired; therefore the voltage function is set to zero applied voltage.

```
TRAMLINE LOAD IMPEDANCE 4.034 1.0 UNIFORM  DISTANCE 1.0 CELLS 11 ;
MATCH vane_slot NEGATIVE NULL LOAD 0.0  0.05 ;
FUNCTION REFLECTED(X)= 0.0 ;
VOLTAGE LOAD 1.0 NEGATIVE REFLECTED ;
```

JOIN Command

Function: Joins two transmission lines together.

Syntax: JOIN primary_line point { POSITIVE, NEGATIVE, SERIES, PARALLEL }
secondary_line point { POSITIVE, NEGATIVE } [CROSSOVER] ;

Arguments:
primary_line - name of primary transmission line, defined in TRAMLINE command.
secondary_line - name of secondary transmission line, defined in TRAMLINE command.
point - spatial coordinate (m).

Description:

The JOIN command is used to connect two transmission lines. The two lines can either be joined at their ends, or the end of one line (the secondary line) can be joined to an interior point of another line (the primary line) to produce parallel or series circuits.

Both transmission lines should have the same cell size at their joining points. The two transmission lines can have different properties at the joint, however. For example, JOIN can be used with two transmission lines of different capacitance and inductance to model impedance mismatch effects.

When the joining point is the end of the transmission line (always the case for the secondary line), the user must specify whether the line lies in the POSITIVE or NEGATIVE direction from the endpoint along the transmission line coordinates. When the joining point is an interior point (primary line only), the user must specify whether the circuit connection is made in SERIES or in PARALLEL. In general, series joints tend to form a voltage divider, while parallel joints tend to form a current divider. Note that if impedance mismatch is to be avoided at series and parallel joints, it is necessary to have a step discontinuity in the impedance function of the primary transmission line. Figure 13-1(a) illustrates the eight possible circuit configurations for end-to-end, series and parallel joints. For illustration simplicity, the transmission lines are depicted as being of the parallel-plate variety.

Eight additional circuit configurations are possible by reversing the voltage polarity of the secondary line with respect to the primary line, as shown in Figure 13-1(b). This is illustrated as a crossover between the top and bottom plates of parallel-plate transmission lines immediately before the joint. These additional eight circuit configurations are invoked with the CROSSOVER option.

Restrictions:

1. All transmission lines must be specified using TRAMLINE commands.
2. For linear and parallel circuits, the line cell sizes must be identical at the joint.
3. The maximum number of JOIN commands is ten.

See Also: MATCH, Ch. 12
TRAMLINE, Ch. 13

Reference:

B. Goplen, J. Brandenburg, and T. Fitzpatrick, "Transmission Line Matching in MAGIC," Mission Research Corporation Report, MRC/WDC-R-102, September 1985.

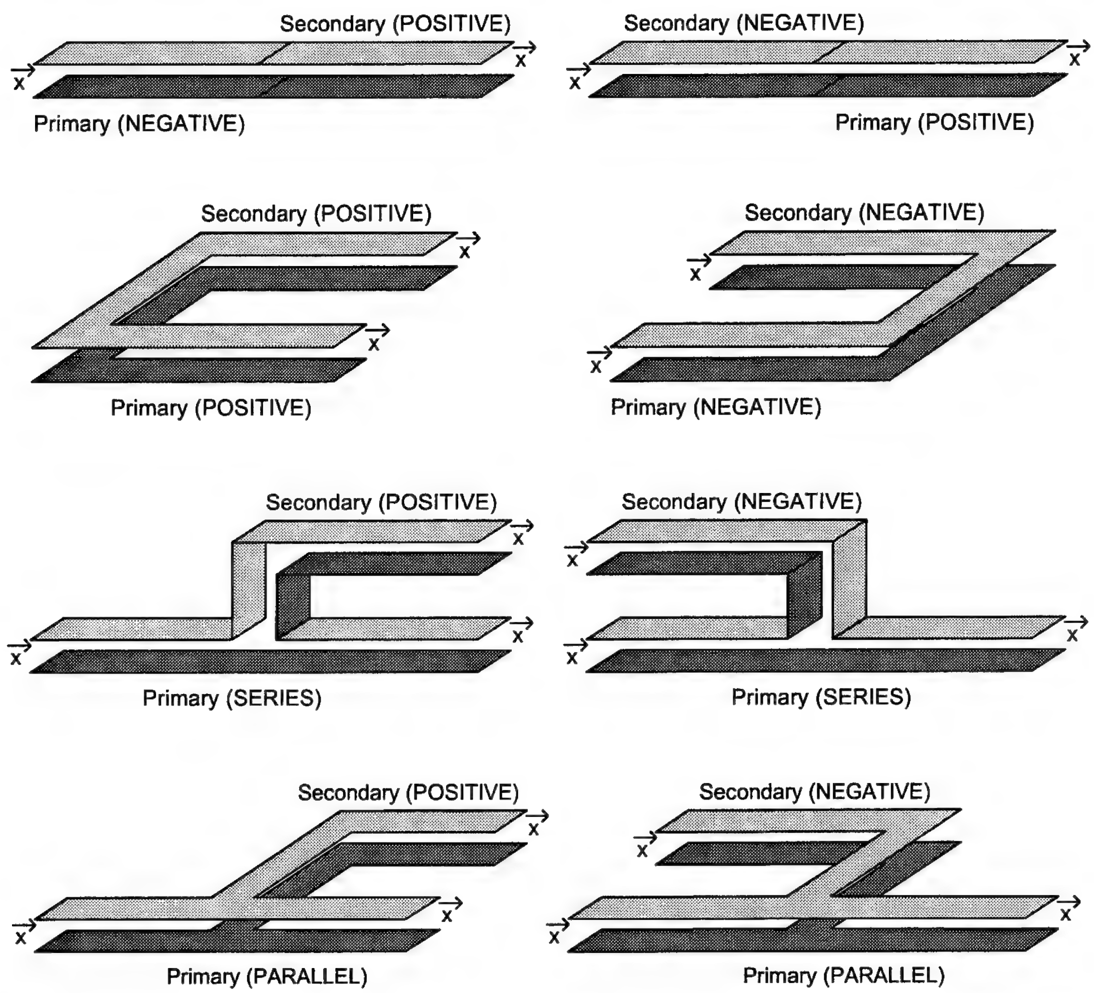


Figure 13-1(a). Default circuit configurations of JOIN with two parallel-plate transmission lines. The sign of the voltage (bottom-plate to top-plate) is preserved between primary and secondary transmission lines.

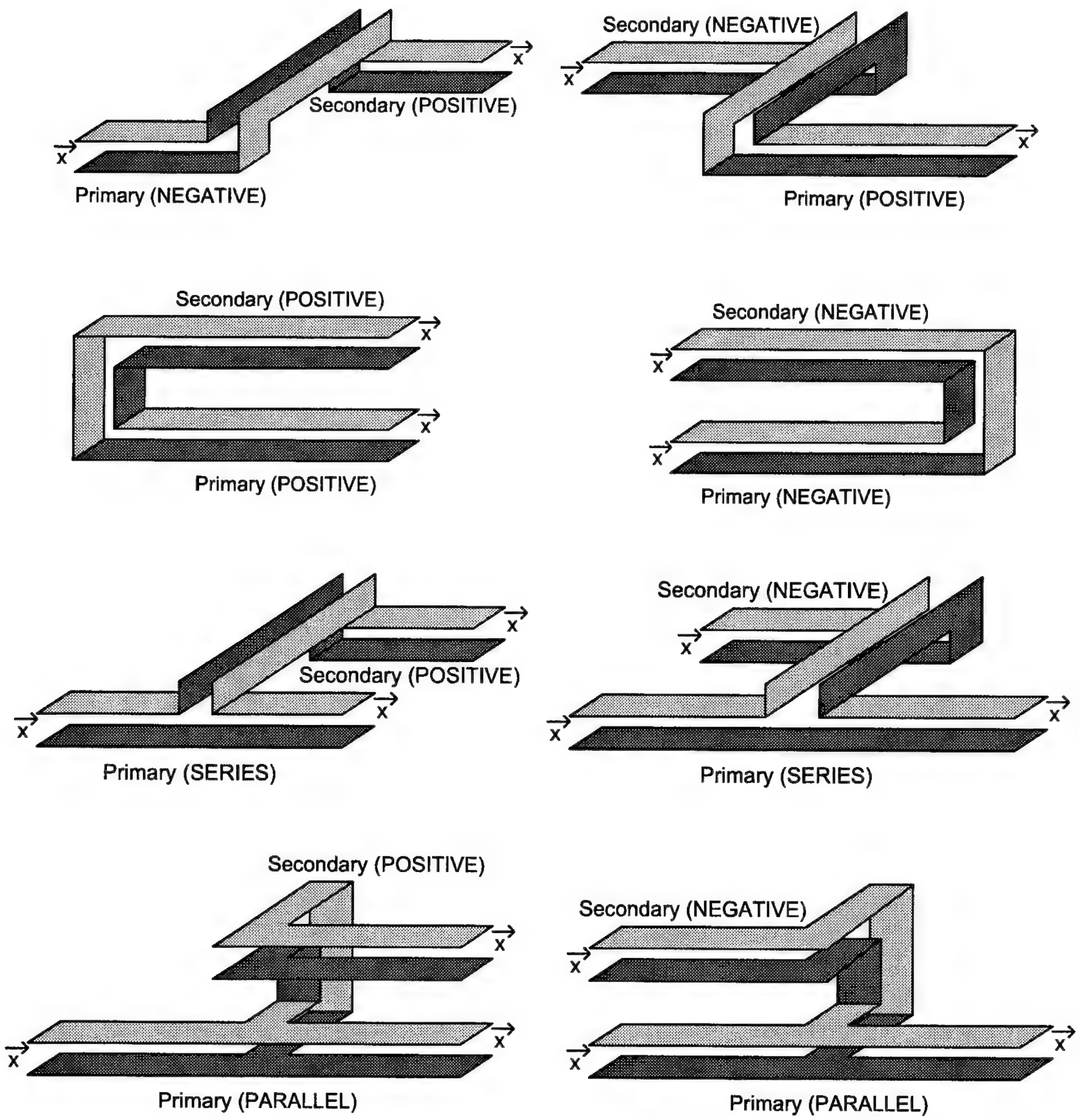


Figure 13-1(b). CROSSOVER circuit configurations for JOIN with two parallel-plate transmission lines. The sign of the voltage (bottom-plate to top-plate) is reversed between primary and secondary transmission lines.

LOOKBACK Command

Function: Specifies a boundary in a transmission line for outgoing waves.

Syntax: LOOKBACK transmission_line point {POSITIVE, NEGATIVE } ;

Arguments:

transmission_line - name of line, defined in TRAMLINE command.
point - spatial coordinate (m).

Description:

The LOOKBACK command specifies an outlet boundary condition which absorbs any outgoing waves from a specified point on the transmission_line. The sense is always from outside the line to inside. If this is in the direction of increasing coordinate, enter POSITIVE; otherwise, enter NEGATIVE.

Restrictions:

1. The total number of LOOKBACK commands in a simulation is limited to five.
2. A FIELDS command must precede any LOOKBACK commands.
3. A Courant criterion limits the time step.

See Also: **VOLTAGE, Ch. 13**

References:

B. Goplen, "Boundary Conditions for MAGIC," Mission Research Corporation Report, MRC/WDC-R-019, presented at the Twenty-third Annual Meeting APS Division of Plasma Physics, 12-16 October 1981.

VOLTAGE Command

Function: Specifies a transmission-line boundary for outgoing (and incoming) waves.

Syntax: VOLTAGE transmission_line point { POSITIVE, NEGATIVE } f(t) ;

Arguments:

transmission_line - name of line, defined in TRAMLINE command.
point - spatial coordinate (m).
f(t) - voltage function, defined in FUNCTION command.

Description:

The VOLTAGE command specifies a voltage function, f(t), which is incoming to a specified point on the transmission_line. The sense is always from outside the line to inside. If this is in the direction of increasing coordinate, enter POSITIVE; otherwise, enter NEGATIVE.

See Also: **FUNCTION, Ch. 6**
LOOKBACK, Ch. 13

Restrictions:

1. The total number of VOLTAGE commands in a simulation is limited to five.

2. A FIELDS command must precede any VOLTAGE commands.

3. A Courant criterion limits the time step.

This page is intentionally left blank.

14. MATERIAL PROPERTIES

This chapter covers the following commands:

CONDUCTANCE	
CONDUCTOR	
DIELECTRIC	
FOIL	(2D simulations only)
VOID	(3D simulations only)
MATERIAL	
SURFACE_LOSS	(3D simulations only)

You can use these commands to assign material properties to spatial objects. (Material properties can be assigned only to areas in 2D simulations and to volumes in 3D simulations.) Material properties can affect both electromagnetic fields and charged particles. In general, particles which enter any spatial object which has been assigned a material property will be destroyed (absorbed on the surface).

The commands which provide explicit material-property effects are CONDUCTANCE, CONDUCTOR, DIELECTRIC, and FOIL. The MATERIAL command provides a list of materials and their properties which can be used by other commands which require the specification of particular material properties (e.g., atomic number, atomic mass, conductivity, density, permittivity, etc).

The VOID command is used to create a void, or vacuum, in a perfectly conducting material. In particular, this command is used in 3D in conjunction with the CONDUCTOR command to build and shape the desired geometry. The conducting elements of the geometry are constructed by assigning to volumes the property of being a CONDUCTOR or a VOID. The operation is performed in the sequence in which the commands appear in the input file. For example, you might start by assigning a large rectangular volume the property of being a CONDUCTOR and then "hollow" out an interior cavity by assigning a cylindrical volume (interior to the rectangular volume) the property of VOID. Thus, by first defining the desired volume's regions of various shapes and then sequentially assigning one of the properties of CONDUCTOR or VOID, you can assemble complicated structures.

The FOIL command is used to insert special thin CONDUCTORs whose physical width is smaller than the cell resolution and allow for the scattering of particles through the foil material. The foil is treated as a perfect conductor electromagnetically, and charged particles which penetrate the foil can undergo multiple scattering with resulting energy loss and even absorption. Particles may be transmitted through the foil or may back scatter out of the incident side of the foil.

MAGIC starts with the entire space being assigned as a vacuum, in other words, VOID. In adding DIELECTRIC or CONDUCTANCE, only regions which are otherwise vacuum can be assigned these properties. Resistivity is also the material property provided by the CONDUCTANCE command; however, in this case, the resistivity is assumed to be known in advance and must be specified as a function of time and space. The CONDUCTOR command provides the extreme case of zero resistivity, or perfect conductivity. It is commonly used to represent metallic components. All electromagnetic fields within the specified volume will vanish identically. Finally, the DIELECTRIC command provides a way to modify the relative permittivity within an area. Where not otherwise specified, the vacuum permittivity of unity will be in effect. By superimposing CONDUCTANCE and DIELECTRIC commands, a lossy dielectric can be simulated.

CONDUCTANCE Command

Function: Applies finite conductivity property to a spatial object.

Syntax:

```
CONDUCTANCE  { area, volume } { material,  sigma(x1,x2,x3) }
        [ SCALE  scale_factor(t) ]
        [ RESONANT  frequency  gamma,  SKIN_EFFECT  frequency ] ;
```

Arguments:

area	- name of spatial area defined in AREA command for 2D simulation.
volume	- name of spatial volume defined in VOLUME command for 3D simulation (conformal volumes only).
material	- name of material, defined in MATERIAL command.
sigma	- conductivity (mhos/m), constant or spatial function defined in a FUNCTION command.
scale_factor	- conductivity scale factor, temporal function defined in a FUNCTION command.
frequency	- center frequency for resonant or skin effect conductivity in Hz.
gamma	- resonance width for resonant conductivity, in 1/sec.

Description:

The CONDUCTANCE command specifies a conductivity (mhos/m) everywhere within the object, which must be an area in a 2D simulation and a volume in a 3D simulation. The conductance may be entered either by specifying a material or by entering conductance as a function of the spatial coordinates. Conductivity will be applied only within the specified spatial object and will result in an additional current as specified by Ohm' Law, $J = \sigma E$.

The SCALE option multiplies the specified conductance by a time-dependent scale_factor. Frequency dependence can also be added with either the RESONANT or SKIN_EFFECT options. The frequency dependence of the RESONANT option is

$$\sigma(\omega) = \sigma_0[1+(\omega^2-\omega_0^2)^2/(\omega\gamma)^2]^{-1},$$

where σ_0 is the specified conductivity, ω_0 is 2π times the specified frequency, and γ is the specified width. The frequency dependence of the SKIN_EFFECT option is approximately

$$\sigma(\omega) = \sigma_0(\omega/\omega_0)^{1/2},$$

within the range $0.5\omega_0$ to $5.0\omega_0$, where ω_0 is 2π times the specified frequency, effectively mimicking the frequency dependence of losses in a skin-depth surface resistivity over one decade in frequency. The frequency dependencies are illustrated in the figures below. Note that both frequency-dependent options also introduce complex parts, e.g., dielectric effects, to the conductivity. (The full complex behavior of resonant absorption is $\sigma(\omega) = \sigma_0(-i\omega\gamma)/(-i\omega\gamma+\omega_0^2-\omega^2)$. The skin effect approximation is achieved by a combination of 2.430 times a frequency-independent conductance minus 1.766 times a resonant absorption with zero frequency and $\gamma=2.236\omega_0$.)

It is useful to note that when conductance is employed over a large area of the simulation, the damping time scale of an oscillatory field in that area will go approximately as $\tau = 2\varepsilon/\sigma$, which corresponds to a $Q = \omega\varepsilon/\sigma$. This formula can be used as a starting point for estimating the desired value of conductivity. However, when conductivity is concentrated in a small area, the relationship is obviously more complicated, and this expression

is not applicable. Also note that the value of σ is related to the desired value of Q, and not to the actual conductivity of the metallic walls, which would be many orders of magnitude too high. A separate model is available in 3D to simulate wall losses (see CONDUCTOR command, Ch. 14). However, in 2D, the only way to simulate wall losses is with a large area of CONDUCTANCE that provides an equivalent Q, as described above.

The conductance field can be output in the same manner as the electromagnetic fields, using the field name SIGMA. The output consists only of the input quantity sigma and does not include the time-dependent scale factor or resonant or skin effects. The ohmic power lost to the conductance material, e.g., the heat generated, can be output by looking at the OHMIC_LOSSES variable with the OBSERVE FIELD_ENERGY command.

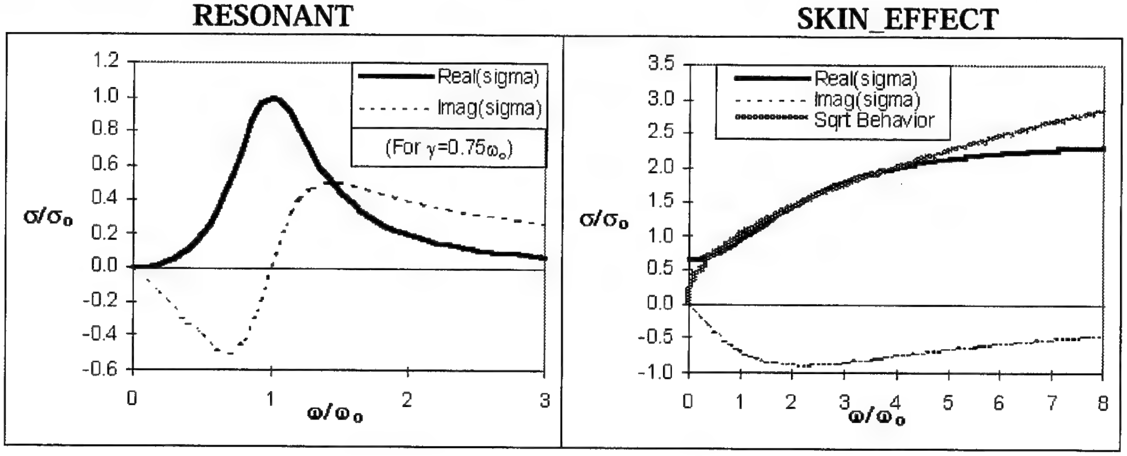
Restrictions:

1. The conductance property can be applied only to areas in 2D simulations and volumes in 3D.
2. The spatial objects must be conformal (in 3D simulations only).
3. The spatial objects must not overlap when using multiple CONDUCTANCE commands.
4. The maximum number of CONDUCTANCE commands that can be specified is 20.
5. The RESONANT option and the SKIN_EFFECT option are available in MAGIC2D only.

See Also: FUNCTION, Ch. 6
MATERIAL, Ch. 14
MAXWELL, Ch. 17
CONDUCTOR, Ch. 14

Reference: *Classical Electrodynamics*, Second Edition, J. D. Jackson, Wiley, New York, 1975, Section 7.5.

RESONANT **SKIN_EFFECT**



Frequency dependence of the RESONANT and SKIN_EFFECT options. SKIN_EFFECT is an approximation to square-root-frequency behavior, valid to 4% within the frequency interval of the shaded region.

CONDUCTOR Command

Function: Assigns perfect (infinite) conductivity property to a spatial object.

Syntax: CONDUCTOR [name]{ area, volume } [MATERIAL material];

Arguments:

name	-	name for conductor.
area	-	name of spatial area defined in AREA command for 2D simulation.
volume	-	name of spatial volume defined in VOLUME command for 3D simulation.
material	-	name of a material, from table or defined in MATERIAL command (default is COPPER).

Description:

The CONDUCTOR command applies perfect (infinite) conductivity everywhere within the spatial object, which must be an area in a 2D simulation and a volume in a 3D simulation.

The CONDUCTOR property is the most basic and most common of the material properties. All electromagnetic fields within perfectly conducting objects vanish. All particles which enter such objects are destroyed on the surface. Furthermore, only those objects which have been assigned this material property may emit particles (EMIT, Ch. 16). The name of a conductor is, by default, the same as that of the object (area or volume) which is being assigned the property of being a conductor. In many cases, however, it is desirable to use a single name for a conductor that may be constructed from multiple pieces. By using the name argument, you may do just that.

The MATERIAL option allows you to specify the conducting material. This is ignored by all of the Maxwell field algorithms (MAXWELL, Ch. 17) that assume that the conductivity is infinite. It is used only by the eigenmode algorithm (EIGENMODE, Ch. 19) to estimate wall losses from a surface current model.

Objects assigned the conducting property are always resolved into conformal segments. The treatment of non-conformal surfaces is different in 2D simulations than in 3D simulations. In 2D simulations, the surface can run diagonally (corner-to-corner) through a cell. (The SHIM model also allows a more accurate treatment of arbitrary surfaces in 2D.) In 3D simulations, the shape is approximated in a stair-step fashion. Once an object has been given the conducting property, portions of the object can be voided (VOID, Ch. 14).

In 2D, the CONDUCTOR command may be used to build the simulation geometry, by adding regions of perfect conductivity. In 3D the CONDUCTOR and VOID commands are used to add and remove regions of perfect conductivity respectively.

See Also:

AREA, Ch. 10
VOLUME, Ch. 10
FOIL, Ch. 14
VOID, Ch. 14
MATERIAL, Ch. 14
SHIM, Ch. 15
EMIT, Ch. 16

Restrictions:

The conductor property can be applied only to areas in 2D simulations and volumes in 3D simulations.

Examples:

1. In a simulation of a magnetron the anode and cathode are constructed of several pieces. Here the cathode is constructed from three objects: cathode, endcap.left, and endcap.right. The anode is constructed from the objects named: anodeshell and vane.1 vane.2 …

```
CONDUCTOR CATHODE ;
CONDUCTOR CATHODE ENDCAP.LEFT ;
CONDUCTOR CATHODE ENDCAP.RIGHT;
CONDUCTOR ANODE ANODESHELL ;
DO I=1,N.VANES;
   CONDUCTOR ANODE VANE.'I' ;
ENDDO;
```

2. In 2D the shape of a collector is created using the AREA command with POLYGONAL shape option. The area is assigned the perfectly conducting property with the CONDUCTOR command, and the resulting assignment may be displayed using the DISPLAY_2D command. The following figure shows the resulting collector.

```
AREA COLLECTOR POLY 5.09E-01 0.00E+00
      5.09E-01 5.59E-02
      2.13E-01 5.59E-02
      1.96E-01 3.81E-02
      1.96E-01 3.56E-02
      2.03E-01 3.56E-02
      2.08E-01 4.06E-02
      4.30E-01 4.06E-02
      4.86E-01 2.54E-03
      4.86E-01 0.00E+00
      5.09E-01 0.00E+00 ;
CONDUCTOR COLLECTOR ;
DISPLAY_2D COLLECTOR MAXWELL ;
```

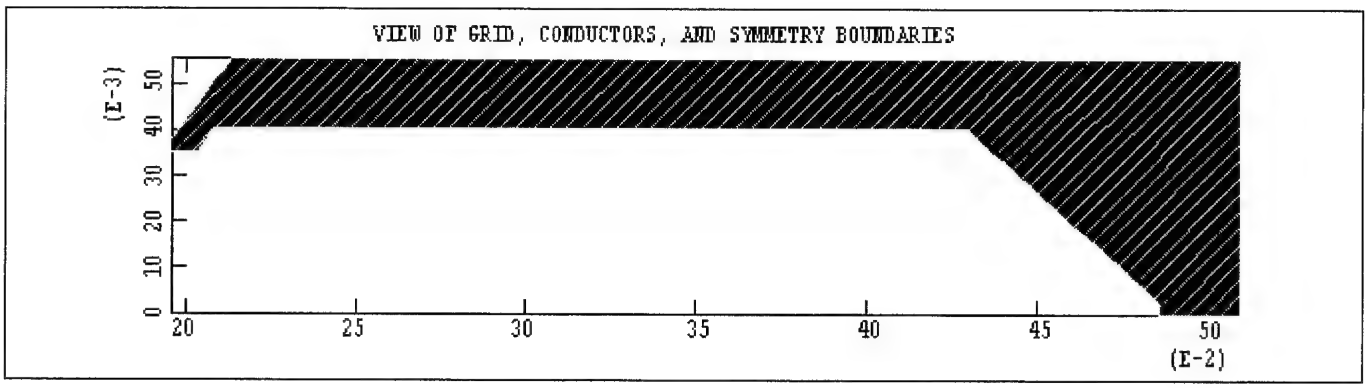


Illustration of the area Collector assigned the property of perfect conductivity.

DIELECTRIC Command

Function: Applies dielectric permittivity property to a spatial object.

Syntax:

```
DIELECTRIC  { area, volume }
        { material, permittivity(x1,x2,x3) }
        [ { EPS1, EPS2, EPS3 } ] ;
```

Arguments:

area	- name of spatial area defined in AREA command for 2D simulation.
volume	- name of spatial volume defined in VOLUME command for 3D simulation (conformal volumes only) .
material	- name of material, defined in MATERIAL command.
permittivity	- relative permittivity (eps/eps0), constant or spatial function defined in a FUNCTION command.

Defaults:

The default permittivity distribution is isotropic; that is, a single scalar permittivity affects all field components equally. To create an anisotropic distribution, you must specify each component of permittivity (EPS1, EPS2, EPS3) independently using separate DIELECTRIC commands (2D simulations only). The default permittivity for unspecified components is unity.

Description:

The DIELECTRIC command applies dielectric permittivity everywhere within the spatial object, which must be an area in a 2D simulation and a volume in a 3D simulation. The permittivity (relative permittivity, also known as the dielectric constant, may be entered either by specifying a material or by entering permittivity as a function of the spatial coordinates. To enter an isotropic dielectric, a single command is sufficient, and no permittivity component is entered. (By default, the permittivity entered will be applied to all three components.) To make the dielectric anisotropic in a 2D simulation, enter separate commands to specify each permittivity component (EPS1, EPS2, EPS3) independently. In 3D only isotropic dielectrics are permitted. As many as three commands may be required, depending on the application. This feature is not available in 3D simulations.

The vacuum value of permittivity is used everywhere else (outside of the spatial object) in the simulation. A particle which strikes the object is destroyed, and its charge is deposited permanently on the surface. If lossy dielectrics are required, conductance can also be applied to the object (CONDUCTANCE, Ch. 14). If the object contains or contacts an outer boundary (e.g., PORT, Ch. 12), then any phase velocity information required in the outer boundary command must account for the dielectric property. (Failure to consider this may result in artificial scattering from the outer boundary.)

The displacement fields (D1, D2, D3) can be output in the same manner as any other fields. The relative permittivity components can also be output using the field names, EPS1, EPS2, and EPS3 in 2D, and using the name DIELECTRIC in 3D.

Restrictions:

1. The dielectric property can be applied only to areas in 2D simulations and volumes in 3D simulations.

2. The spatial objects must be conformal (in 3D simulations only).

3. Anisotropic dielectrics require one or more DIELECTRIC commands to be entered. This feature is available only in 2D simulations.

4. Use of a DIELECTRIC command may alter the phase velocity and thus affect input requirements for outer boundary commands.

5. The maximum number of DIELECTRIC commands that can be entered is twenty.

See Also: **PORT, Ch. 12**
MATERIAL, Ch. 14
CONDUCTANCE, Ch. 14

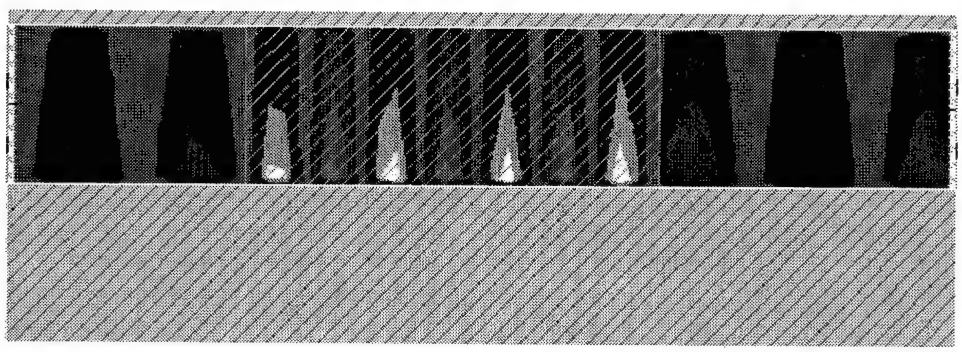
Examples:

1. An anisotropic dielectric with a relative permittivity of four in the X2 component and unity for the other two components can be created with the command,

```
DIELECTRIC Gap   4   EPS2     ;
```

2. In this 2D simulation example, an RF wave is propagating down a coax 60 cm in length with inner and outer radii of 10 cm and 20 cm respectively. A dielectric plug of length 26.25 cm is at the center of the coax with a relative permittivity of four. Notice that the wave in the dielectric is compressed and appears to have a wavelength of half that in the vacuum region of the coax. The commands to insert the dielectric are:

```
AREA RINGINSERT CONFORMAL 15CM,10CM,41.25CM,20CM ;
DIELECTRIC RINGINSERT  4  ;
```



Magnetic field contours of an RF wave propagating down a coax with a dielectric insert.

FOIL Command

Function: Assigns a thin foil property to a spatial object. The thin foil property consists of perfect conductivity and of electron transport through the foil. Electron transport is achieved by using the ITS model for electron scattering dynamics in materials. This includes forward and backward scattering and absorption of electrons.

Syntax:

FOIL { area, volume } thickness
{symbol, ALLOY name density nelements symbol fraction, symbol fraction, ... }
[ENERGY_LIMITS energy_min energy_max]
[EXIT_SPECIES species]
[RECORD timer [{ BINARY, ASCII }]];

Arguments:

area	- name of conformal area, defined in AREA command. (2D only)
volume	- name of conformal volume, defined in VOLUME command (3D only).
thickness	- thickness of foil (meters).
symbol	- 1- or 2-letter atomic symbol of an element, e.g., Al, Fe, Cu, Ni, Au, etc.
name	- name of alloy, arbitrary
density	- alloy mass density (kg/m^3)
nelements	- number of elements in alloy.
fraction	- fraction of the element (by weight) in alloy.
energy_min	- lowest allowable electron energy in the foil (MeV) default is the maximum of 0.001 MeV or 1/244 of the maximum energy, below this energy electrons are deposited within the foil.
energy_max	- maximum anticipated electron energy (MeV). ITS uses this as an upper limit for the scattering cross section tables. (Default value of 1.0 MeV.)
species	- species of electron after exiting the foil, may be different from incident to distinguish incident and exiting electrons, defined in species command.
start	- the time to start recording particle data (seconds).
stop	- the time to stop recording particle data (seconds

Description:

The FOIL command is used to simulate a thin, perfectly conducting sheet of material in a simulation and the scattering of electrons as they pass through such a foil. The foil's simulation area or volume <u>must be exactly one cell thick</u> in the direction of transit. The actual physical thickness of the foil is assumed to be much smaller than a cell width and is input as a separate command argument.

Any particle with the electron charge-to-mass ratio, which penetrates the foil, will experience scattering, energy loss, and possible back scattering or deposition within the foil. This complicated process is modeled by subroutines from the Integrated TIGER Series of codes (ITS 3.0). The scattering parameters are automatically calculated from the material properties and the thickness of the foil. The foil appears as a perfect conductor to the electromagnetic fields, so that electric fields parallel to its surface vanish. Also, particles not having the electron charge-to-mass ratio, e.g., ions are destroyed on the foil in exactly the same manner as for a perfect conductor.

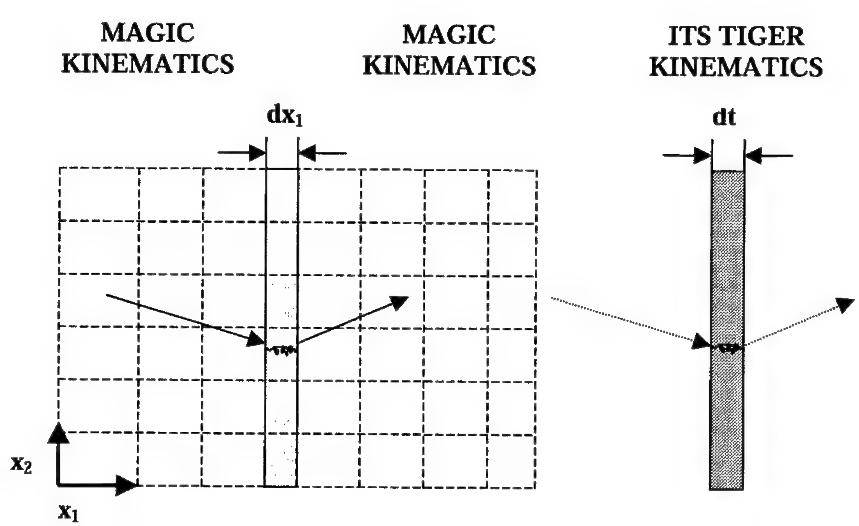
The default foil material in MAGIC is aluminum, atomic symbol Al. You may specify any element using the MATERIAL option and the one- or two-letter atomic symbol for that element. The choice of material affects only the electron scattering attributes of the foil. MAGIC treats all materials as perfect conductors, regardless of

the actual electrical conduction properties of the selected element. It is also possible to specify an alloy material such as stainless steel, instead of a single element. In this case, you must supply a name for the alloy, the mass density of the alloy, density, the number of elements in its composition, nelements, and a list of the atomic symbol for each element, together with its weight fraction in the alloy. The sum of the weight fractions must be unity.

The ENERGY_LIMITS option is used to specify the electron energy limits for the scattering model in the ITS model. The first argument, energy_min, specifies the minimum energy of an electron which is tracked in the foil without being absorbed. This argument is used by ITS's CYLTRAN code unit. Electrons that fall below this energy are destroyed in the foil. The default value of the minimum energy is a maximum of 5 keV or 1/244 of the maximum electron energy. The second argument, energy_max, specifies the maxium anticipated energy of an electron entering the foil. It is used by ITS's XGEN program as an upper bound for the scattering cross-section and energy attenuation tables. The default value of the maximum energy is 1 MeV. If an electron enters the foil with an energy above the anticipated maximum, it will automatically be reassigned the maximum energy.

Normally, an electron exits the foil as the same species that entered the foil. It is, however, possible to distinguish incident electrons from scattered electrons by changing the species identification of the electron when it exits the foil, e.g., from RED_ELECTRONS to BLUE_ELECTRONS, if those two species have been created with the SPECIES command. This feature is expected to be of use when electrons are incident from both sides of a foil, or when it is desired to distinguish the incident from back-scattered electrons. The EXIT_SPECIES option invokes this feature. All particles with the electron charge-to-mass ratio will exit the foil as the specified species, regardless of its identity when incident on the foil.

Use of the FOIL command requires the presence of the ITS code, "XGEN.EXE", and the XGEN data file "FORT.9" in the same directory as the MAGIC executable. After all the FOIL commands are entered, MAGIC automatically constructs an input file for XGEN, which it then executes. XGEN creates a "FORT.11" output file containing the scattering cross-section tables, which are used by the CYLTRAN subroutines embedded in MAGIC.



This figure illustrates electron kinematics for FOILs. Particles to the left or right of the foil use the standard MAGIC kinematics based on the Lorentz force. Particles which enter the foil are passed to the ITS Tiger kinematics subroutines, where they experience collisions with the atomic nuclei. When foils are range-thick a significant reduction in the electron energy can occur, and in some cases the electrons can be deposited in the foil rather than exiting as either scattered or back-scattered electrons.

For each FOIL command, the RECORD option creates two files that contain information about the particles that are incident on the foil and those particles that exit from the foil. The default file format is ASCII. The names of the files have the form foilname_record_in.dat and foilname_record_exit.dat, where foilname is the name of the foil. Each of the record files begins with the simulation time step, SYS$DTIME, and a table for the species, followed by the particle information. A new line is added for each particle that is incident on the foil or exits from the foil. There are no headers or titles in the files. The files only contain the actual particle data. The data structure of the file is as follows:

```
SYS$DTIME
numSpecies
iSpecies species_name q_over_m m_species
iSpecies species_name q_over_m m_species
...
iSpecies species_name q_over_m m_species
iTimeStep iSpecies charge x1 x2 x3 p1 p2 p3
iTimeStep iSpecies charge x1 x2 x3 p1 p2 p3
iTimeStep iSpecies charge x1 x2 x3 p1 p2 p3
...
iTimeStep iSpecies charge x1 x2 x3 p1 p2 p3
```

For the ASCII data option, the Fortran-style formats are:

```
Sys$dtime: ()
NumSpecies: ()
species table: (i4,1x,a32,1x,e13.6,1x,e13.6)
particle table: (i9,1x,i4,7(1x,e13.6))
```

For BINARY, all values are four bytes long except the species_name, which is a character value with a length of 32 bytes.

References:

"ITS Version 3.0: The Integrated TIGER Series of Coupled Electron/Photon Monte Carlo Transport Codes," by J. A. Halbleib, R. P. Kensek, T. A. Mehlhorn, G. D. Valdez, S. M. Seltzer, and M. J. Berger, SAND91-1634 (March 1992). Also Oak Ridge National Laboratory document CCC-467.

Incorporation of ITS with MAGIC is as follows. The XGEN program is used, in situ, to generate scattering cross section and energy attenuation tables for the foil material. A subset of the CYTRAN code unit is linked with the MAGIC executable, and is used to evaluate the transport of electrons through the foil. Mission Research Corporation expresses its appreciation to the Radiation Shielding Information Center (RSIC) at Oak Ridge National Laboratory and the ITS development team for granting permissions to use XGEN and CYLTRAN in this manner.

Restrictions:

1. The area or volume must be conformal.

2. The maximum number of foils is 127.

3. XGEN.EXE and the FORT.9 data file must be in the same directory as the MAGIC executable.

See Also: CONDUCTOR, Ch. 14.

Examples:

The following example illustrates the scattering of an electron beam by a gold foil. The gold foil is specified in the simulation by using the following commands.

```
zlo=-0.5*dz ; zhi=+0.5*dz ;
foilThickness = 8microns ; ! 1/50th range at 1 MeV
AREA thinwall CONFORMAL zlo, 0.0, zhi, 0.975m ;
MARK thinwall X1 SIZE dz;
SPECIES redElectron CHARGE -1 MASS 1 ELECTRON ;
FOIL thinwall foilThickness GOLD ENERGY 0.001 1.0
        EXIT_SPECIES redElectron RECORD ASCII 0.0 1.0;
```

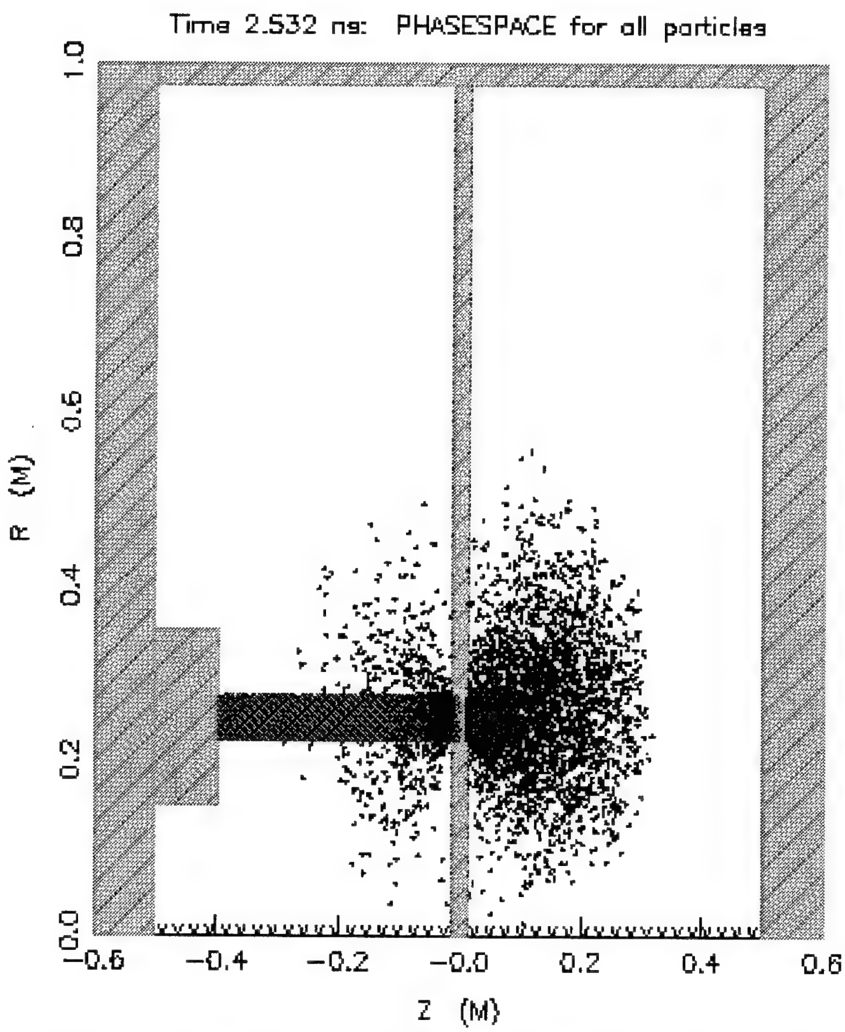


Figure of an energetic electron beam scattered by a thin gold foil.

VOID Command

Function: Assigns void property to a spatial object.

Syntax: VOID { volume } ;

Arguments: volume - name of spatial volume defined in VOLUME command for 3D simulation.

Description:

The VOID command applies a voiding operation everywhere within a specified spatial object. It is used only in conjunction with the perfectly conducting property (CONDUCTOR, Ch. 14). Specifically, use a CONDUCTOR command to make an object perfectly conducting and then use a VOID command to remove the conducting effect from some specified portion of the object. Normally, the voided object will overlap some portion of the original conducting object (See Examples, below). This process of assigning conducting and void properties can be continued ad infinitum, making it possible to create a conducting structure of great complexity.

See Also: **VOLUME, Ch. 10**
CONDUCTOR, Ch. 14

Restrictions:

The VOID property can be applied only to conducting volumes in 3D simulations. It is not implemented in 2D.

Examples:

In this 3D simulation, we create a perfectly conducting, cubic object and then cut out the lower left-hand quarter.

```
Volume  Cube  Conformal     -10 cm, -10 cm, -10 cm,
                             10 cm,  10 cm,  10 cm;
Volume  Hollow  Conformal   -9 cm, -9 cm, -9 cm
                            +9 cm, +9 cm, +9 cm;
CONDUCTOR  cube ;
VOID  hollow ;
```

MATERIAL Command

Function: Specifies material properties.

Syntax:

```
MATERIAL material
        [ ATOMIC_NUMBER atomic_number ]
        [ ATOMIC_MASS mass_number ]
        [ MOLECULAR_CHARGE charge_number ]
        [ MOLECULAR_MASS mass_number ]
        [ CONDUCTIVITY sigma ]
        [ MASS_DENSITY density ]
        [ PERMITTIVITY epsilon ]
        [ IONIZATION_PARAMETERS m2 c ]
        [ LIST ] ;
```

Arguments:

material	- material name, user-defined.
atomic_number	- atomic number.
charge_number	- molecular charge number (Σ Z).
mass_number	- mass number (amu).
density	- density (kg/m**3).
sigma	- conductivity (1/ohm-m).
epsilon	- relative dielectric constant (unitless).
m2, c	- constants used in ionization cross-section calculation (unitless).

Defaults:

Some material properties have already been entered and are ready for use. However, any other properties that are required must be entered explicitly.

Description:

An internal material table contains the properties of materials that may be used in the simulation. You can add other material properties or change existing properties using MATERIAL commands.

The MATERIAL command associates a set of physical properties with an arbitrary material name. Other commands that require values for these properties may reference the name specified in the MATERIAL command. The ATOMIC_NUMBER, ATOMIC_MASS, MOLECULAR_CHARGE, MOLECULAR_MASS, CONDUCTIVITY, MASS_DENSITY, PERMITTIVITY and IONIZATION_PARAMETERS options specify particular material properties. You can use these options to enter a physical value into the table, or you can list all of the properties of a specified material using the LIST option. (If you specify ALL instead of a material name, the entire table will be printed.)

The MOLECULAR_CHARGE, MOLECULAR_MASS and IONIZATION_PARAMETERS options specify values that are used when the IONIZATION command is used. The IONIZATION_PARAMETERS option specifies values for M^2 and C which are used to calculate the ionization cross section, σ_{ion}, using

$$\sigma_{ion} = 0.5\ [\ h^2 / (mc\beta)^2\ ]\ [\ M^2\ (\ 2\ \ln(\gamma\beta) - (\beta^2) + C\ ]$$

where $0.5\ [\ h^2 / (mc)^2\ ] = 1.874 \times 10^{-20}\ \text{cm}^2$.

The ATOMIC_MASS, ATOMIC_NUMBER and MASS_DENSITY properties are used when the FOIL command is used. There are no predefined materials for FOIL.

The CONDUCTIVITY property is used when the CONDUCTOR ... MATERIAL option is used.

The PERMITTIVITY property can be used with the DIELECTRIC command.

See Also:

FOIL, Ch. 14
IONIZATION, Ch. 16
CONDUCTOR, Ch. 14
DIELECTRIC, Ch. 14

The material properties currently defined are shown in the following tables.

Material Conductivity

Material	σ (mhos/m)
ALUMINUM	3.72E+07
BRASS	1.56E+07
COPPER	5.99E+07
GOLD	4.10E+07
GRAPHITE	71400
IRON	1.03E+07
MOLY	1.75E+07
MONEL	2.38E+06
NICHROME	1.00E+06
NICKEL	1.45E+07
PLATINUM	9.52E+06
SILVER	6.17E+07
SOLDER	7.04E+06
STAINLESS	1.38E+06
TUNGSTON	1.82E+07
VACUUM	0

Material Permittivity

Material	κ ($\varepsilon/\varepsilon_0$)	Material	κ ($\varepsilon/\varepsilon_0$)
AIR	1	QUARTZ_FUSED	3.8
BAKELITE	4.8	RUBBER	3.2
CERAMIC_ALSIMAG_393	4.95	RUBY_MICA	5.4
ETHYL_ALCOHOL	25	SANDS	4
GLASS_CORNING_707	4	SILICA_FUSED	3.8
ICE	4.2	SODIUMCHLORIDE	5.9
NEOPRENE	3.4	STYROFOAM	1.03
NYLON	4	TEFLON	2.1
PAPER	3	TIO2	100
PLEXIGLASS	2.9	DISTILLED_WATER	80
POLYETHYLENE	2.25	SEA_WATER	20
POLYSTYRENE	2.55	VACUUM	1
PORCELAIN	6	WATER	80

Material Ionization Constants

Material	Molecular Charge Number+	Molecular Mass (amu)	M^2	C
HYDROGEN_GAS	2	2	0.695	8.12
HELIUM	2	4.003	.77	7.7
ARGON	18	39.948	4.2	37.9
MERCURY	80	200.59	5.7	64
WATER_VAPOR	10	18	3.2	32.3
NITROGEN_GAS	14	28	3.74	34.8
OXYGEN_GAS	16	32	38.8	38.8
NEON	10	20.18	2.0	18.2
KRYPTON	36	83.8	6.1	52.4
XENON	54	131.3	8.0	72.4
CARBON_DIOXIDE	22	44	5.75	57.9

Restrictions:

The maximum number of entries in the material table is five hundred.

SURFACE_LOSS Command

Function: Turns on the model to compute surface losses for conductors. (3D only)

Syntax: SURFACE_LOSS frequency ;

Arguments: frequency - the frequency for evaluating the surface losses, in Hz.

Description:

The SURFACE_LOSS command specifies that the simulation will include the effects of conductor losses due to surface currents. Evaluation of the surface losses requires the computation of a skin depth, which in turn, requires both frequency and material conductivity information. The present model assumes a single fixed frequency as supplied by the user in this command. Material conductivity information is based upon the use of the MATERIAL option in the CONDUCTOR command. When a material is not specified in the CONDUCTOR command, copper is assumed.

The user can also use the OBSERVE FIELD_POWER command, with SURFACE_LOSS as the power variable, to display the surface losses within an arbitrary volume of the simulation.

The computation of surface losses during time-domain simulation is accomplished by attenuating the B field at the surface at each time step, based on the effective physical equations for conductor surface current losses. No attempt to model the behavior of the fields within the metal is made. Specifically, a surface consisting of a non-ideal metal possesses a non-zero, parallel electric field that is proportional to the parallel magnetic field. When this non-zero parallel electric field is accounted for in Faraday's Law, it results in the addition of an effective magnetic conductivity term for the parallel magnetic field at the surface, e.g.,

$$\partial_t \mathbf{B}_{surface} = -\nabla \times \mathbf{E}_{normal} - \eta \mathbf{B}_{surface} ,$$

where the factor η contains the skin depth information. The user should be aware that this surface loss model is sometimes susceptible to appreciable finite-difference error effects so that the model's computed surface losses may depart from true behavior by as much as 15%. The principal source of error is the stair stepping approximation to curved surfaces made by the grid.

See Also: **CONDUCTOR, Ch. 14**
OBSERVE, Ch. 22

Examples:

The following example illustrates the use of SURFACE_LOSS in conjunction with "OBSERVE FIELD_POWER SURFACE_LOSS object" to include surface losses and display a plot of the surface power dissipation. The plot displayed by OBSERVE will show the instantaneous power dissipated at the surface of any conductors within the volume named "cavity." The commands are:

```
SURFACE_LOSS 4.0GHz ;
OBSERVE FIELD_POWER SURFACE_LOSS Cavity ;
```

15. UNIQUE GEOMETRY

This chapter covers the following commands:

CABLE	(3D simulations only)
CAPACITOR	(3D simulations only)
DAMPER	(3D simulations only)
DIPOLE	(3D simulations only)
POLARIZER	(2D simulations only)
SHIM	(2D simulations only)
STRUT	

You can use these commands to assign unique geometry properties to spatial objects. (Points, lines and areas are the only spatial objects which can have a unique geometry property associated with them.) Whereas material properties affect both electromagnetic fields and charged particles, unique geometry properties generally affect only the fields. Particles which cross the spatial object are unaffected, except through the fields (i.e., there is no absorption).

The commands listed above invoke different geometry effects. Their unifying feature is that they all represent physical effects that occur on a spatial scale too small to be resolved by any realistic spatial grid. As a class, they have been described as "sub-grid" models. It is because of this small spatial scale that these models are applied only to lines and areas (which have zero thickness).

The POLARIZER and SHIM commands are available only in 2D simulations. The CABLE, CAPACITOR, DAMPER, and DIPOLE commands are available only in 3D simulations. The STRUT command is available in both 2D and 3D simulations.

The POLARIZER command approximates a three-dimensional structure, specifically, an infinite array of wires. For example, in two-dimensional cylindrical coordinates, it can effectively represent a three-dimensional helical structure. It is assigned only to a line object in a 2D simulation. In non-trivial applications, the model couples transverse magnetic and transverse electric field components which are generally independent in a purely electromagnetic, two-dimensional simulation. For conformal orientations, the model can block either field component while letting the other component pass.

The SHIM command produces a very fine-scale geometric variation on the surface of a conductor. It is assigned only to a line object in a 2D simulation. (The scale of the variation must be smaller than the surrounding spatial grid.) This model can be used to represent a very gradual slope or a rippled effect on a surface.

The CABLE command represents electromagnetic excitation at a conducting surface, such as would be caused by a small-scale cable stub or loop. It is assigned only to a point object in a 3D simulation. The cable stub is presumed to penetrate the conducting surface through a co-axial conductor and to terminate in vacuum. It radiates as a dipole antenna. The cable loop is similar, but terminates on the conducting surface. It radiates as a quadrupole antenna. It is applied to a spatial line one cell in length at the conducting surface.

The CAPACITOR command represents the capacitive effect of a small-scale gap. It is assigned only to an area object in a 3D simulation. (It is applied to an area one cell deep between two adjacent conductors.)

The DAMPER command represents the resistive effect of a membrane damper, a very thin resistive sheet used to damp out electric fields in a specified plane in space. (This technique is commonly used in electromagnetic cavities to damp out undesirable electromagnetic modes.) It is assigned only to an area object in a 3D simulation.

The DIPOLE command is used to represent the effect of a dipole moment on a conducting surface. Physically, this could result from high-intensity photoemission that creates a space-charge-limited boundary layer. If the e-folding distance associated with the layer cannot be easily resolved by the spatial grid, then this approximation may be applicable. It is assigned only to an area object in a 3D simulation. (It is applied to an area on the conducting surface.)

The STRUT command represents the inductive, capacitive, and resistive properties of a small-scale wire, or strut. It is applied to a spatial line in either 2D or 3D simulations, but is presently limited to conformal orientations.

CABLE Command

Function: Applies coaxial cable excitation to a spatial point in a 3D simulation.

Syntax: CABLE point { X1, X2, X3 } current(t) { STUB stub_length, LOOP loop_area };

Arguments:

point	- spatial coordinates or name of point defined in POINT command.
current	- temporal function for cable current (A), defined in FUNCTION command.
stub_length	- length of antenna stub (m).
loop_area	- area of antenna loop (sq. m).

Description:

The CABLE command applies electromagnetic excitation at a single spatial point. It represents the effects of a stub or loop antenna on the surface of a conducting object.

The STUB option represents a short extension of the center conductor which projects out of a cable embedded in a conducting object. The stub is assumed to be aligned with a coordinate axis (x1, x2, or x3) which is normal to the conducting surface, and the current is applied over the stub_length to excite the electric field at that point.

The LOOP option represents a small loop formed by the center conductor which projects out of and bends back into the conducting object, thus forming a loop. The surface normal to the loop area is assumed to be aligned with a coordinate axis (x1, x2, or x3) which is tangential to the conducting surface, and the current is applied over the periphery of the loop_area to excite the magnetic field at that point.

Restrictions:

1. The spatial point must be immediately above the conducting surface.
2. Only 100 CABLE commands may be entered.
3. This command is available only in 3D simulations.

See Also: POINT, Ch. 10

CAPACITOR Command

Function: Simulates capacitance between two conducting objects in a 3D simulation.

Syntax: CAPACITOR area { X1, X2, X3 } capacitance ;

Arguments:

area	- name of spatial area, defined in AREA command.
capacitance	- capacitance per unit length (farads/meter).

Description:

The CAPACITOR command represents a capacitive effect between two adjacent conducting objects (or adjacent sections of the same conducting object). Generally, the dimensions associated with the actual capacitor will be much smaller than the spatial grid; hence, brute-force resolution of the effect is not possible.

To use this sub-grid model, the gap between the conducting objects must be exactly one cell in depth. Use of this command, therefore, requires control of the spatial grid, so that the area containing the gap (designated by the area name) must be one cell in depth. The direction associated with this "depth" must be specified as a coordinate axis (X1, X2, or X3). Finally, you must specify the capacitance per unit length — the capacitance effect is in the specified direction (between the objects), and the length is measured transversely (along the gap).

Restrictions:

1. The spatial area designated by the area name must be exactly one cell in depth; however, the actual length of the cell does not matter to the correct application of the capacitive effect.

2. Only 100 CAPACITOR commands may be entered.

3. This command is available only in 3D simulations.

See Also:

AREA, Ch. 10
AUTOGRID, Ch. 11
GRID [options], Ch. 11

DAMPER Command

Function: Applies a membrane damper to an area object in a 3D simulation.

Syntax: DAMPER area { material, conductivity } thickness ;

Arguments:

area	-	name of spatial area, defined in AREA command.
material	-	name of material, defined in MATERIAL command.
conductivity	-	membrane conductivity (mhos/m).
thickness	-	membrane thickness (m).

Description:

The DAMPER command simulates a thin membrane of electrically conducting material which is applied over the specified spatial area. The area must be conformal. Only components of the electric field which are parallel to and lie within the plane of the area will be attenuated. The membrane conductivity can be specified either by entering the name of a material or by entering conductivity directly. The membrane resistivity will be calculated from

$$\rho = \frac{1}{\sigma t}$$

where σ is the conductivity and t is the thickness.

Restrictions:

1. The area object must be conformal.
2. Only 50 DAMPER commands may be entered.
3. This algorithm functions only in time-domain simulations (not EIGENMODE).
4. This command is available only in 3D simulations.

See Also: AREA, Ch. 10
MATERIAL, Ch.14

DIPOLE Command

Function: Applies a dipole drive to the surface area of an object in a 3D simulation.

Syntax: DIPOLE area { POSITIVE, NEGATIVE } photon_source ;

Arguments:
area - name of spatial area, defined in AREA command.
photon_source - name of the photon source, defined in PHOTON command.

Description:

The DIPOLE command applies a time-varying electric dipole on the surface area designated by the area name. It simulates the effect of photoemission on the surface of a conducting object. Although the dipole model is not strictly geometric (it represents space-charge rather than geometrical structure), it is included in this Chapter because it is a classic example of a sub-grid model, i.e., it represents a physical effect too small to be resolved by the spatial grid.

The choice of POSITIVE or NEGATIVE indicates the sense of the surface. If an escaping electron would move in the same direction as the coordinate, enter POSITIVE. Otherwise, enter NEGATIVE. The photon_name identifies the photon source, which can provide both a temporal and a spatial distribution with respect to a localized photon source. The dipole moment is then given by

$$\partial_t Q_1 = s(t - r / c)\, \chi(r)$$

where s is the function of retarded time and χ is the geometric effect of photon spreading. The radius, r, represents the distance from the photon source to any point on the photoemission surface.

Restrictions:

1. The spatial area must be a conformal surface of a spatial object.
2. This command is available only in 3D simulations.

See Also:
AREA, Ch. 10
PHOTON, Ch. 16

POLARIZER Command

Function: Applies a polarizer geometry in a 2D simulation.

Syntax: POLARIZER line pitch_angle(x1,x2) [algorithm] ;

Arguments:

line	-	name of conformal line, defined in LINE command.
pitch_angle	-	pitch angle (degrees), constant or function defined in a FUNCTION command.
algorithm	-	end-point algorithm (-1, 0, or +1).

Description:

The POLARIZER command provides a capability to model certain three-dimensional conducting structures in a 2D simulation. In effect, it couples TE and TM electromagnetic fields to reflect the correct conditions for an array of "wires" at an angle to the axis of symmetry. A perfect application is a helical geometry in cylindrical coordinates. In this case, the helix sheath approximation described by Pierce (see References) provides an excellent theoretical discussion. There are other, non-helical applications as well, including wave-splitters, antenna shields, and waveguide screens.

The polarizer is applied along a single, conformal line. The pitch_angle (in degrees) describes the orientation of the wires in the array, relative to the X3 (ignorable) axis, measured in the positive direction of the X1 axis or the X2 axis. A functional specification allows the pitch angle to vary with the relevant spatial coordinate.

The treatment of the polarizer endpoints can be controlled via the algorithm. In general, the endpoints of the polarizer may simply float in space or they may be terminated on other boundaries, such as a conductor, an antenna, or an outgoing model. The centered algorithm (algorithm=0) does not impose the polarizer conditions at the endpoints so the conditions of any terminating boundary will still be met despite the presence of the polarizer. Two other algorithms apply the polarizer condition at one end, either at the right/upper endpoint (algorithm=+1) or at the left/lower endpoint (algorithm=-1). These two choices have the important property that they are exactly energy conserving for all wavelengths, whereas the centered algorithm can result in a small degree of damping on the shortest wavelengths. In situations where this damping may affect physical results, such as for a coarsely gridded helix, the energy-conserving algorithms should be used; however, for well-resolved wavelengths, there is no appreciable difference between algorithms.

Particles pass through a polarizer boundary unscathed. Of course, particle trajectories may be affected by strong electromagnetic fields in the vicinity of the boundary.

Restrictions:

1. A maximum of 50 polarizers are allowed.
2. The line must be conformal.
3. The polarizer is intended to be an internal object, and is not suitable for an outer boundary condition.

See Also:

OUTGOING, Ch. 12
CONDUCTOR, Ch. 14
MODE, Ch. 17
DRIVER, Ch. 19

References:

J. R. Pierce, *Traveling Wave Tubes*, Van Nostrand Company, Princeton, NJ, 1950, Appendix 2.

D. Smithe, G. Warren, and B. Goplen, "A Helix Polarization Model for 2-D Particle-in-Cell Simulation of Helix Traveling Wave Tubes," APS Div. of Plasma Physics, Seattle, WA, November 1992.

B. Goplen, D. Smithe, K. Nguyen, M. Kodis, and N. Vanderplaats, "MAGIC Simulations and Experimental Measurements from the Emission Gated Amplifier I & II Experiments," 1992 IEDM, San Francisco, CA, December 1992.

Examples:

In the following example (cylindrical coordinates), a model of a helix is created between axial coordinates, ZI and ZF, at a radius coordinate, RI. The pitch angle (PSI_DEGREES) is measured from the azimuthal coordinate (ϕ) toward the axial coordinate (z). The polarizer is given the name "HELIX."

```
LINE HELIX_LINE CONFORMAL ZI RI ZF RI ;
POLARIZER HELIX PSI_DEGREES ;
```

SHIM Command

Function: Applies a thin, perfectly conducting shim in a 2D simulation.

Syntax: SHIM CONDUCTOR area THICKNESS thickness(x) WINDOW area ;

Arguments:

area	- name of spatial object, defined in AREA command.
thickness	- shim thickness, constant or defined in FUNCTION command.

Description:

The SHIM command is used to add a small, perfectly-conducting layer, or shim, to a conformal surface of a perfectly conducting object. The thickness of the layer must be smaller than the spatial cell size. Thus, this command provides a means to model a fine-scale variation in a conducting surface without resolving the variation with the spatial grid.

The shim is applied to a conformal surface of a conducting area. This spatial object must be defined in an AREA command. Furthermore, the spatial object must also be perfectly conducting (CONDUCTOR, Ch. 14). The thickness must be a fraction of the cell size and may be constant or a function of one of the spatial coordinates. The precise location of the shim is provided by the WINDOW area, which must enclose a segment of one of the conformal surfaces that forms the perimeter of the conducting object. In other words, it selects where on the conducting object the shim is to be applied.

The conformal surface where the shim is applied can have a length as short as one spatial cell, or it can be much longer. It is possible to taper or to modulate such an extended surface by means of the thickness function. However, the thickness is not allowed to exceed the height of the cell adjacent to the surface. A Courant violation may occur if this constraint is not observed. Care should also be taken to avoid using the shim in the immediate presence of certain types of outer boundaries. A good practice is to terminate the thickness function at both ends with zero thickness and to stay well removed from port and outgoing wave boundaries.

Restrictions:

1. The maximum number of SHIM boundaries that can be specified is fifty (50).

2. The shim model can be applied only to conformal surfaces of perfectly conducting area objects.

3. Non-uniform spacing in the direction normal to the conducting surface is not allowed (the first two cells must be of equal size).

4. The thickness of the shim must not exceed the height of the cell adjacent to the conducting surface.

See Also:

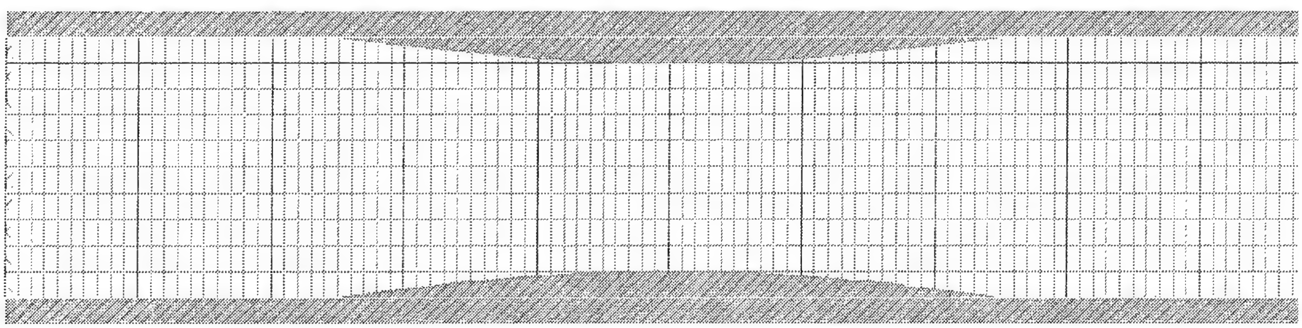
AREA, Ch. 10
CONDUCTOR, Ch. 14
FUNCTION, Ch. 16
TIME_STEP, Ch. 17
MAXWELL, Ch. 17

Examples:

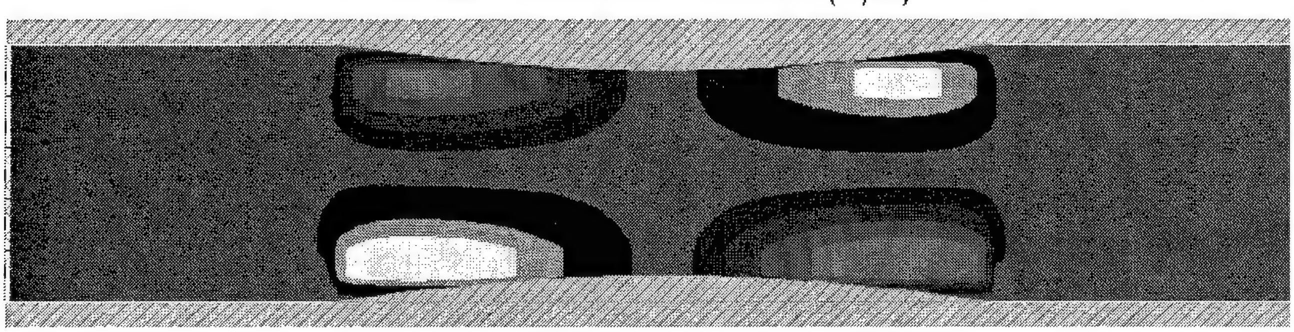
In the following test case, we define a coaxial cable with large radii and known impedance. An incident, semi-infinite wave is introduced and propagates through the cable without reflection. Next, a shim is introduced on the anode (case shown below) and another on the cathode. The thickness varies sinusoidally from zero to a maximum value and then to zero. The following figures illustrate the resulting geometry and grid and show the effect of the impedance change in the coax on the axial electric field.

```
XI = 25.0CM ; XIP = XI + 5*DELTA.X1;
XF = 75.0CM ; XFP = XF - 5*DELTA.X1;
PK = 3.14159/(XF-XI) ;
FUNCTION Th(X1) = STEP(X1,XI)*STEP(XF,X1)*DELTA.X2*SIN( (X1-XI)*PK ) ;
X1I = XI ; X2I = SYS$X2MN+0.99*DELTA.X2 ;
X1F = XF ; X2F = SYS$X2MX-0.99*DELTA.X2 ;
AREA CLIP.SHIM CONFORMAL X1I,X2I X1F,X2F ;
SHIM CONDUCTOR ANODE   THICKNESS Th  WINDOW CLIP.SHIM ;
SHIM CONDUCTOR CATHODE THICKNESS Th  WINDOW CLIP.SHIM ;
```

VIEW OF GRID, CONDUCTORS, AND SYMMETRY BOUNDARIES



Time 9.435 ns: Of Ez (V/m)



These figures illustrate the alteration of the coaxial radii by using shims, and the introduction of an axial Ez field near the impedance mismatch.

STRUT Command

Function: Applies a strut (inductive circuit element) geometry in a 2D or 3D simulation.

Syntax:

```
STRUT  line strut_diameter(x1,x2,x3)
       [ INDUCTANCE  inductance(x1,x2,x3) ]
       [ { MATERIAL  material  frequency ,
           RESISTIVITY  resistivity(x1,x2,x3)
           RESISTANCE  resistance(x1,x2,x3) } ]
       [ NUMBER strut_number ]
       [ SUBCYCLES  sub_cycles ] ;
```

Arguments:

line	- name of conformal line, defined in a LINE command.
strut_diameter	- strut diameter (meters), constant or spatial function defined in a FUNCTION command.
inductance	- name of forced inductance function (henrys/meter), defined in a FUNCTION command.
material	- name of material, specified in MATERIAL command.
frequency	- frequency for the material surface resistivity (hertz).
resistivity	- name of strut surface resistivity function (ohms), defined in a FUNCTION command.
resistance	- name of forced resistance function (ohms/meter), defined in a FUNCTION command.
strut_number	- number of struts in third dimension of 2D simulation (struts/meter in cartesian and polar or struts around axis in cylindrical and polar).
sub_cycles	- number of sub-cycles (default is 1).

Description:

The STRUT command provides a capability to model certain three-dimensional conducting elements in a 2D or 3D simulation. Such elements are generally thin conducting rods or "struts" which permit current to flow along them, often connecting one conductor to another. In 2D, the struts typically are widely spaced in the simulation's ignorable direction, allowing electromagnetic wave energy at all but very low frequencies to pass through; thus, they cannot be represented as a solid conductor, since this would effectively block the passage of all electromagnetic energy. In the simulation, the strut must lie on a conformal line designated by the line name.

The principal physical attribute of the strut is the generation of strong magnetic fields near its surface, where current flows, leading to a behavior essentially identical to an inductive circuit element. The inductance is determined primarily by the diameter of the rod, which is specified by the function, strut_diameter. In cases where a circular cross section is not sufficient, or when precise control of the inductance is required, the user has the option of directly specifying the inductance per length in henrys/meter with the INDUCTANCE option. If this option is entered, the strut_diameter value is not used to calculate inductance.

A strut can also have a resistive component. By default, the resistance of a strut is zero. There are three means provided to assign resistance to a strut. The most simple means is with the MATERIAL option, which accepts a material name and a frequency in order to compute the surface resistivity (or "resistance per square," = 1/(conductivity*skin-depth)). A second option, RESISTIVITY, allows you to specify a value of surface resistivity for a material that may not be present in the existing table (MATERIAL, Ch. 14). The resistance of the rod is then calculated from the surface resistivity and the strut_diameter. As with the inductance, you also

have the option of specifying the resistance per length directly in ohms/meter with the assistance of the RESISTANCE option.

In 2D simulations, you must also use the NUMBER option to specify the strut_number in the ignorable (third) dimension.

It is known that the strut model causes an additional Courant-limit constraint on the time step and may additionally require sub-cycling of the strut model. In general, sub-cycling will be necessary when the time step exceeds the centered-difference Courant limit, as is possible with the time-biased and High-Q MAXWELL options; the number of sub-cycles as specified in the SUBCYCLES option is expected to go as the ratio of the time step to the centered-difference limit. The reduction of the maximum allowable time step is usually very small in 2D and somewhat larger in 3D, becoming noticeable for very small inductances, e.g., when the rod diameter approaches the length of the rod or the distance between rods. The exact strut-affected courant limit for the time step is currently unknown, so the user may need to use trial and error techniques to find a suitably small time step which does not result in a courant instability.

STRUT variables which can be output with the OBSERVE command are CURRENT, CHARGE, INDUCTANCE, RESISTANCE, STORED_ENERGY, and OHMIC_LOSS.

Restrictions:

1. A maximum of ten struts are allowed.

2. The line must be conformal to the grid.

3. The line may not contact an object which has a dielectric permittivity property.

4.The number of sub-cycles is the same for all struts and is taken to be the value of ncycles from the last strut command to use the SUBCYCLES option.

See Also: **CONDUCTOR, Ch. 14**
DIELECTRIC, Ch. 14
MATERIAL, Ch. 14
TIME_STEP, Ch. 17
DIAGNOSE, Ch. 21
OBSERVE, Ch. 22

Example:

The following 2D example illustrates a wide gap cavity with six, 1.5-millimeter-diameter struts across the gap. The cavity is intended for use with a very intense beam propagating close to the drift tube wall. Without the space charge and return current on the drift tube wall, the beam would space-charge limit and fail to propagate. The strut's inductive property causes it to appear as a metal conductor at low frequencies while simultaneously being transparent to high frequencies. Thus, the strut carries the steady-state beam return current and image charge, so that the beam can propagate across the gap. However it also passes the high-frequency cavity oscillation, allowing the cavity to interact with beam bunching and image charge, so that the beam can propagate across the gap. However it also passes the high-frequency cavity oscillation, allowing the cavity to interact with beam bunching.

```
SYSTEM CYLINDRICAL ;
AREA TUBE_AND_CAVITY POLYGON 0.,RTUBE
      ZBGN_GAP,RTUBE              ZBGN_GAP,RIN_CAVITY
      ZBGN_CAVITY,RIN_CAVITY      ZBGN_CAVITY,ROUT_CAVITY
      ZEND_CAVITY,ROUT_CAVITY     ZEND_CAVITY,RIN_CAVITY
      ZEND_GAP,RIN_CAVITY         ZEND_GAP,RTUBE
      ZEND,RTUBE ;
LINE GAP CONFORMAL ZBGN_GAP,RTUBE ZEND_GAP,RTUBE ;
STRUT GAP 6 1.5_MM ;
```

This page is intentionally left blank.

16. EMISSION PROCESSES

This chapter covers the following commands:

EMISSION [options]	
EMISSION BEAM	
EMISSION EXPLOSIVE	
EMISSION GYRO	
EMISSION HIGH_FIELD	(2D simulations only)
EMISSION PHOTOELECTRIC	
EMISSION SECONDARY	(3D simulations only)
EMISSION THERMIONIC	(2D simulations only)
EMIT	
PHOTON	
IONIZATION	(2D simulations only)

You can use these commands to describe the emission of charged particles from the surfaces of an object.

Most of the parameters required for the emission models are specified using default values. You can change any of these values using the EMISSION [options] command. You can use the rest of the EMISSION commands to create customized models of the following emission processes:

BEAM	- an arbitrary, prescribed beam.
EXPLOSIVE	- field-extraction from a surface plasma.
GYRO	- a prescribed beam for gyro devices.
HIGH_FIELD	- high-field (Fowler-Nordheim) emission.
PHOTOELECTRIC	- photoelectric emission.
SECONDARY	- secondary emission.
THERMIONIC	- thermionic emission.

The first three are "artificial" emission processes (as opposed to fundamental processes that involve the work function). In BEAM emission, the beam properties are assumed to be known a priori and are simply prescribed, without regard for the underlying fundamental processes. For example, it can be used to model a beam that enters a cavity, without including the processes that create the beam in the first place. In EXPLOSIVE emission, one or more species is extracted from an assumed surface plasma, a phenomenon typically encountered in pulsed-power applications. The underlying processes of high-field emission and joule heating, which produce the plasma, are considered only phenomenologically. The surface plasma is assumed, but not actually modeled with particles. Instead, particles are created with charge sufficient to cause the normal electric field at the surface to vanish. This zero, or near-zero, surface field is a distinguishing feature of EXPLOSIVE emission. GYRO emission is really a special case of BEAM emission in which the controls have been adapted specifically for gyro (rotating beamlet) applications.

The last four represent fundamental emission processes, in which energy is supplied to overcome a work function. In HIGH_FIELD emission, the energy is supplied by the ambient electric field and results in quantum-mechanical tunneling. The electron yield thus varies with the electric field on the surface according to the Fowler-Nordheim equation. In PHOTOELECTRIC emission, the energy to overcome the work function is supplied by an incident photon. The energy spectrum of the emitted electrons depends on the incident photon spectrum, and a transport code is typically used to compute the electron energy spectrum, which is required input for this emission model. The spatial distribution of the photon flux is specified with a PHOTON command, used

only in conjunction with PHOTOELECTRIC emission. In SECONDARY emission, the energy to overcome the work function is supplied by energetic charged particles that are incident upon a conducting surface. The electron yield varies with the material and the angle and energy of the incident particle. In THERMIONIC emission, the energy to overcome the work function is thermal, and the electron yield varies with the surface temperature, according to the Richardson-Dushman equation.

Once an emission model has been created, you can enable emission on a particular object by using the EMIT command. In 3D simulations, the EMIT command can be used only with volume objects (VOLUME, Ch. 10) and not from points, lines, or areas. In 2D simulations, the EMIT command can be used only with area objects (AREA, Ch. 10). In both cases, the spatial objects must be perfectly conducting (CONDUCTOR, Ch. 14).

The following table presents some general guidelines on the selection of emission algorithms. Alternatives are best used only with a good understanding of the algorithm and the underlying physics. For example, EXPLOSIVE emission is recommended for Child-Langmuir applications, since the current density is critically dependent upon the surface dynamics (normal electric field and charge density). An attempt to use BEAM emission in such applications will either under-emit, producing a non-zero surface field, or over-emit, causing low-velocity particles to pile up at the surface, which prevents resolution. It is unlikely that the proper Child-Langmuir dynamics will be obtained in either event.

General conditions for algorithm selection.

EMISSION PROCESS	BEAM	EXPLOSIVE	GYRO	HIGH_ FIELD	PHOTO-ELECTRIC	SECONDARY	THERM-IONIC
Magnetic Insulation		X					
Child Langmuir		X					
Diodes		X					
Field-emitter Arrays				X			
Prescribed Beams	X						
Prescribed Gyro-beams			X				
Beam Formation		X			X	X	X
Multi-factor						X	
Thermionic Cathodes							X

EMISSION [options] Command

Function: Specifies options which may be applied in any emission process command.

Syntax: EMISSION process arguments

```
[ MODEL model ]
[ SPECIES species ]
[ NUMBER creation_rate(t,x1,x2,x3) ]
[ TIMING step_multiple ]
[ SURFACE_SPACING { RANDOM, UNIFORM } ]
[ OUTWARD_SPACING { RANDOM, FIXED } displacement(t,x1,x2,x3) ] ;
```

Arguments:

process	-	emission process (BEAM, EXPLOSIVE, GYRO, HIGH_FIELD, PHOTOELECTRIC, and THERMIONIC).
arguments	-	arguments for specific emission processes.
model	-	name of emission model, user-defined.
species	-	particle species name (ELECTRON, PROTON, or defined in SPECIES command).
creation_rate	-	particle creation rate (particles/cell/time step), constant or defined in FUNCTION command.
step_multiple	-	LORENTZ time-step multiple (integer).
displacement	-	maximum displacement outward from the surface (m), constant or function defined in FUNCTION command.

Defaults:

The following option default values are set. If they are adequate for your requirements, no changes are needed. If you wish to use different values, simply enter the desired keywords and new values when you enter the EMISSION process command.

model	-	set to the name of the process (e.g., BEAM, etc.).
species	-	ELECTRON.
creation_rate	-	3 particles / cell / emission time step.
step_multiple	-	1 Lorentz time_step / emission time step.
displacement	-	initial_velocity x step_multiple x SYS$DTIME.

The default values for both spacing options (SURFACE_SPACING and OUTWARD_SPACING) are set to RANDOM. (For GYRO emission, the SURFACE_SPACING default is UNIFORM.)

Description:

The various emission processes (BEAM, EXPLOSIVE, etc.) have some common control options, all of which are set to default values. These common features are described here, while those features unique to individual emission processes are described in the commands that immediately follow.

The emission model name allows you to specify a unique name for each model that you create. This name will be referred to in EMIT commands. The default model name is the same as the process (e.g., BEAM, EXPLOSIVE, etc.). However, if you create more than one model using the same process, then you must give each model an individual, unique model name.

The species name can be used to specify ELECTRON or PROTON, which is permanently stored in the internal species table. If another choice is desired, the desired particle parameters must be specified in a SPECIES command. The default species name is ELECTRON.

The creation_rate is the number of discrete particles that will be created per cell at each emission time step. Note that this has nothing to do with the amount of charge created, but is strictly a statistical parameter. The data entry can be an integer constant or it can be a function that allows greater control of particle statistics in space and time. The default value is three.

The emission time step is expressed as an integer step_multiple of the LORENTZ (not MAXWELL) time step. The default value is unity.

The SURFACE_SPACING option allows you to specify precisely how particles will be spaced going along the surface. Its main effect is on the way trajectories appear in plots. The default value is RANDOM, which provides a physical appearance. As an alternative, the selection of UNIFORM generally produces continuous striations, which may aid in interpreting phase-space output without adversely affecting the physics.

The OUTWARD_SPACING option allows you to specify precisely how particles will be spaced going outward (away from the surface). Usually, it is best not to position particles precisely on the surface, but to give them some small, outward displacement. (This option can have an important effect on the solution.) Two choices are available for distributing particles within this displacement. The default is RANDOM, which spaces particles randomly within the displacement distance. This choice generally provides superior charge continuity, making maximum statistical value of the particles. The alternative is FIXED, which positions all particles precisely at the outermost extent of the displacement. The default value for displacement equals the product of the particle velocity, the Lorentz time step, and the emission step_multiple.

Restrictions:

Up to 20 EMISSION commands of all types may be used in a simulation.

See Also:

FUNCTION, Ch. 6
EMISSION BEAM, Ch. 16
EMISSION EXPLOSIVE, Ch. 16
EMISSION GYRO, Ch. 16
EMISSION HIGH_FIELD, Ch. 16
EMISSION PHOTOELECTRIC, Ch. 16
EMISSION THERMIONIC, Ch. 16
EMIT, Ch. 16
LORENTZ, Ch. 18
SPECIES, Ch. 18

EMISSION BEAM Command

Function: Specifies a beam emission model.

Syntax:

```
EMISSION  BEAM  current_density(t,x1,x2,x3)  beam_voltage(t,x1,x2,x3)
          [ THERMALIZATION  fraction(t,x1,x2,x3) ]
          [ COSINES  x1cos(t,x1,x2,x3)  x2cos(t,x1,x2,x3)  x3cos(t,x1,x2,x3) ]
          [ options ] ;
```

Arguments:

current_density - incident current density (A/sq. m), constant or function defined in a FUNCTION command.
beam_voltage - beam voltage (eV), constant or function defined in a FUNCTION command.
fraction - thermalization fraction, constant or function defined in a FUNCTION command.
x1cos, etc. - directional cosines, constant or function defined in a FUNCTION command.
options - see EMISSION [options] command.

Defaults:

None, except for the option defaults. The EMISSION BEAM command must be entered for this model to be used.

Description:

The incident current_density for beam emission is described by the equation,

$$\frac{d^2q}{dAdt} = J(t, x_1, x_2, x_3)$$

which allows the current density (A/m2) to depend on time, t, and the spatial coordinates. The initial kinetic energy and momentum for each particle are computed relativistically from the beam_voltage,

$$T = qV(t, x_1, x_2, x_3)$$

which is also allowed to be a function of time and the spatial coordinates. In the absence of a transformation with the COSINES option, the momentum will be directed outward, normal to the emitting surface.

For obvious reasons, no defaults are provided for these two functions. Therefore, both current_density and beam_voltage must be entered in an EMISSION BEAM command before this model can be used.

The options provide the possibility of thermalizing the beam or giving it a direction (which is not directed normally outward) from the surface. The THERMALIZATION option allows the specified fraction of beam energy to be distributed in the transverse coordinates. The energy in each coordinate is determined randomly, and the dominant (normal) component is re-normalized to preserve the specified beam_voltage. The COSINES option causes the beam to propagate in a direction not normal to the emitting surface. The directional cosines are applied to the momentum computed from the beam_voltage to obtain the non-normal components. Preservation of the specified beam_voltage is not checked; therefore, care should be taken that the directional cosines supplied are a unitary transformation.

Restrictions:

Up to 20 EMISSION commands of all types may be used in a simulation.

See Also: FUNCTION, Ch. 6
EMISSION [options], Ch. 16
EMIT, Ch. 16

References:

B. Goplen and R. Worl, "Numerical Simulations of a Monotron Oscillator Cavity Traveling Wave Tube," Mission Research Corporation Report, MRC/WDC-R-098, July 1985.

F. Friedlander, A. Karp, B. Gaiser, J. Gaiser, and B. Goplen, "Transient Analysis of Beam Interaction with Antisymmetric Mode in Truncated Periodic Structure Using 3-Dimensional Computer Code SOS," Trans. IEEE, ED-2, Vacuum Electron Devices, November 1986.

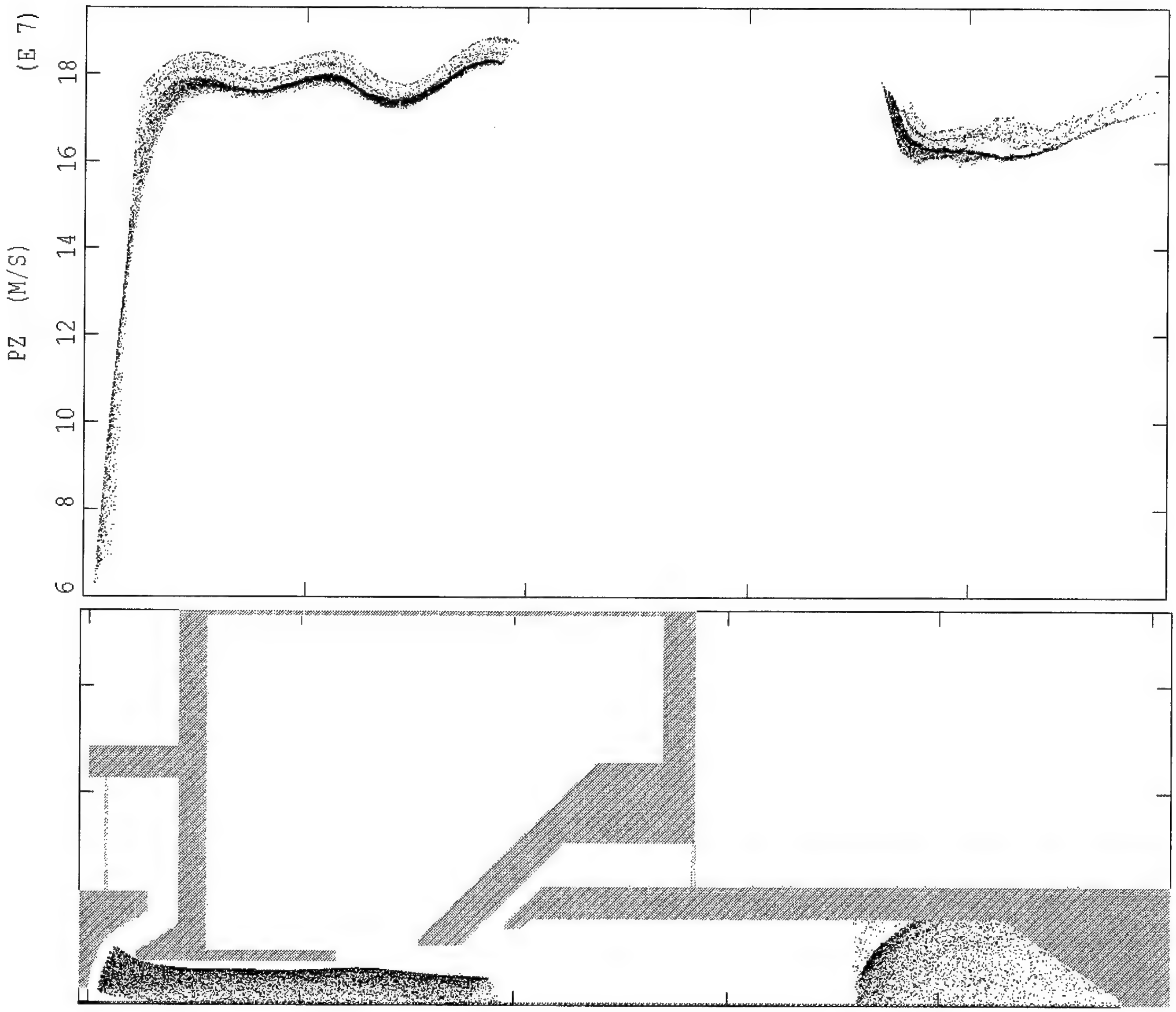
B. Goplen, R. Worl, W. M. Bollen, and J. Pasour, "Vircator Modulation Study," Mission Research Corporation Report, MRC/WDC-R-139, November 1987.

Examples:

This 2D example illustrates the use of beam-emission to simulate a gated emission gun. The klystrode amplifier is an axisymmetric power tube that uses a gated emission Pierce gun to supply the electron beam. In this example, a spherical cathode surface is represented using a non-uniform spatial grid. The EMISSION BEAM command is used with a bunch shape function to simulate the emission properties of the gun. The Pierce gun is controlled with an external RF circuit driven at the desired bunching frequency. Here, the function, SHAPE, is used to modulate the electron beam, which is emitted only during the positive half of the RF cycle. Note that inclusion of the step function in SHAPE limits the beam to a single bunch.

```
! Define Class B bunch shape function and define beam voltage

OMEGA = 2.PI*FREQUENCY ;
PEAKJ = 0.968E4 ;    ! units are A/m**2
PERIOD = 1.0/FREQUENCY ;
FUNCTION Current_SHAPE(T)=PEAKJ*MAX(SIN(OMEGA*T),0.0)*STEP(T,PERIOD) ;
BEAM_ENERGY = 71 ;          ! corresponds to 5x106 m/sec
EMISSION BEAM Current_SHAPE BEAM_ENERGY ;
EMIT BEAM CATHODE ;
```

The figures on the following page illustrate creation of the electron bunch using these commands. Once the electrons are created, they are accelerated by an applied voltage pulse applied at the anode-cathode gap (dashed line). This can be seen by examining the top figure which gives the pz versus z phase space. The cathode and focusing electrodes provide the basic beam profile. The figure also illustrates the resonator cavity used for extracting RF from the bunched beam and the collector for capturing the spent beam. These figures are early in the simulation time, prior to the extraction cavity having reached saturation.



These figures show the early-time electron beam bunches in a klystrode. The upper figure shows the axial momentum phase space versus time. The lower figure shows the geometry. It includes the electron gun, the resonator, and the collector.

EMISSION EXPLOSIVE Command

Function: Specifies an explosive-emission process.

Syntax:

```
EMISSION EXPLOSIVE
        [ THRESHOLD threshold_field(t,x1,x2,x3) ]
        [ RESIDUAL residual_field(t-tb,x1,x2,x3)
        [ PLASMA formation_rate(t-tb,x1,x2,x3) ]
        [ MINIMUM_CHARGE particle_charge(t-tb,x1,x2,x3) ]
        [ MAXIMUM_CURRENT_DENSITY current_density(t-tb,x1,x2,x3) ]
        [ options ] ;
```

Arguments:

threshold_field - breakdown field threshold (V/m), constant or defined in FUNCTION command.
residual_field - electric field residual (V/m), constant or defined in FUNCTION command.
formation_rate - plasma formation rate (0<f<1), constant or defined in FUNCTION command.
particle_charge - minimum particle charge (coul), constant or defined in FUNCTION command.
current_density - maximum current density (A/sq. m), constant or defined in FUNCTION command.
options - see EMISSION [options] command.

Defaults:

In addition to the option defaults, the following additional defaults are employed.

threshold_field - $2.3X10^7$ V/m.
residual_field - 0 V/m.
formation_rate - $f = t/\tau$ ($t<\tau$) or 1 ($t>\tau$), with $\tau = 5x10^{-9}$ sec.
particle_charge - 0 coul
current_density - ∞ A/m^2.

If all of the default values are adequate for your requirements, simply enter EMISSION EXPLOSIVE. The EMISSION EXPLOSIVE command must be entered for this model to be used. If values other than the defaults are desired, include the relevant keywords and new values.

Description:

Explosive emission results from plasma formation on a material surface. Almost any surface exhibits microscopic protrusions, or "whiskers." When exposed to large voltages, electric field enhancement at the whiskers can cause significant high-field emission (quantum-mechanical tunneling overcoming the work function). Subsequently, the whisker may dissipate due to Joule heating, resulting in the formation of a plasma on the material surface. This surface plasma will typically "emit" under the influence of the ambient electric field, with the species extracted from the plasma being determined by the sign of the field.

Our model largely ignores the physical details of the plasma formation process, relying instead on a phenomenological description. However, the particle emission itself is based upon Child's law of physics, specifically, the normal electric field vanishing at the plasma surface.

In our phenomenological treatment of plasma formation, breakdown can occur only if the normally directed field at the half-cell, E_c, exceeds some specified breakdown field_threshold, or $|E_c| > E_{threshold}$.

This test is performed continuously for every surface cell on the emitting object. If the field at a particular cell exceeds the threshold, then that cell is said to "break down." (A single, non-emitting cell between two emitting cells is also allowed to break down, even if the threshold is not exceeded.) The time of breakdown, t_b, is recorded for each cell that breaks down. Subsequently, every cell has its own history and is treated independently.

Once initiated by breakdown, plasma formation in a cell is assumed irreversible. However, the cell does not become fully effective instantly; instead, plasma formation is assumed to occur gradually, so that the effectiveness of the cell increases from zero to unity according to a user-specified formation_rate. (The default formation_rate is linear with a 5-nsec rise time.) For many applications you may wish to control the plasma fraction. Setting the plasma fraction to $f(t) = 1\text{-}\exp(-(t/t_{rise})^2)$, where t_{rise} is ≥ 2 time the voltage rise time equation generally gives good response and emission performance.

It is important to realize that we do not actually create a surface plasma using particles (an approach which leads to poor results in many applications). Instead, we use our phenomenological model and Gauss's law to calculate the charge that would be drawn away from the surface. Thus, a cell that has broken down may "emit" charged particles due to the influence of the ambient electric field. If the field is of one sign, electrons will be emitted; if the other sign, then protons (or positively charged ions) will be emitted. (Note that each species must be enabled in separate EMISSION and EMIT commands.) The calculation is based upon application of Gauss's law to the half-cell, allowing for a small residual field at the surface,

$$\frac{dq}{dA} = \varepsilon_0 f(t - t_b)(E_c - E_r) - \rho\, dx$$

where f is the plasma formation_rate (note the dependence on t_b), and ρ is the existing charge density at the surface (which accounts for incoming as well as outgoing particles of all species).

Additional restrictions may be placed on the created particles in the form of maximum current_density and minimum particle_charge, according to

$$\left|\frac{dq}{dAdt}\right| < J_{maximum}$$

and

$$|dq/n| > Q_{minimum} \quad .$$

All of the explosive emission arguments allow either a constant or a function to be entered. Functions can depend on time, t, in seconds and the spatial coordinates, x1, x2 and x3. Note that only the threshold_field is a function of absolute, or simulation, time. The other four arguments (if functions) are always referenced to the time of cell breakdown, which generally varies with location on the surface.

Restrictions:

1. Up to 20 EMISSION commands of all types may be used in a simulation.

2. The initial_velocity and initial_distance SPACING option should never both be set to zero, since this will cause particles to become trapped on the surface.

See Also: **FUNCTION, Ch. 6**
EMISSION [options], Ch. 16
EMIT, Ch. 16

References:

R. B. Miller, *An Introduction to the Physics of Intense Charged Particle Beams*, Plenum Press, 1982.

B. Goplen, R. E. Clark, and S. J. Flint, "Geometrical Effects in Magnetically Insulated Power Transmission Lines," Mission Research Corporation Report, MRC/WDC-R-001, April 1979.

D. B. Seidel, B. C. Goplen, and J. P. Van Devender, "Simulation of Power Flow in Magnetically Insulated Convolutes for Pulsed Modular Accelerators," presented at the 1990 Fourteenth Pulse Power Modulator Symposium, June 3–5, 1980.

B. Goplen and R. Worl, "Pulsed Power Simulation Problems in MAGIC," Mission Research Corporation Report, MRC/WDC-R-124, February 1987.

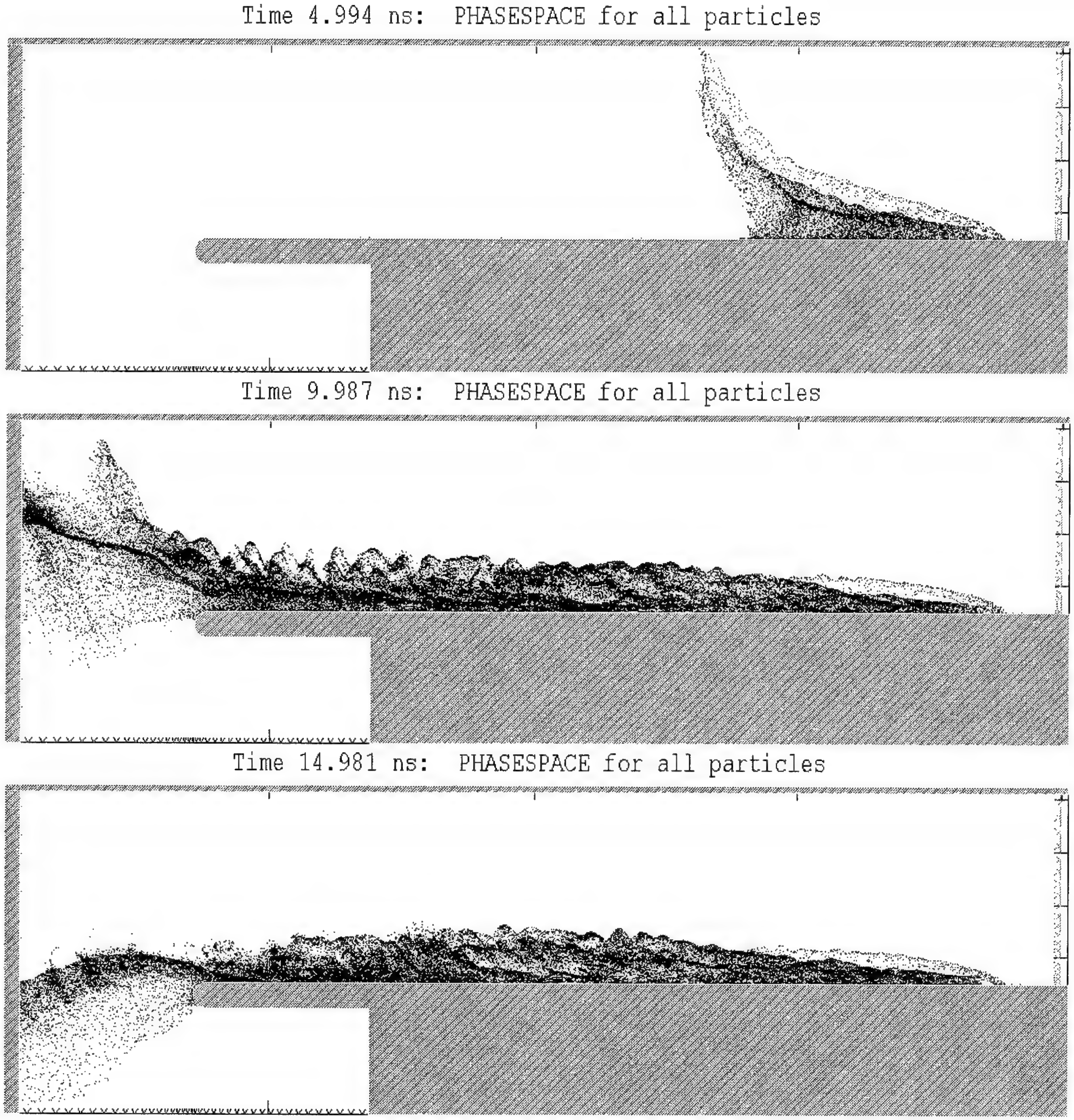
B. Goplen, R. Worl, J. McDonald, and R. Clark, "A Diagonal Emission Algorithm in MAGIC," Mission Research Corporation Report, MRC/WDC-R-152, December 1987.

Examples:

This 2D example illustrates explosive emission from the Aurora cathode. It uses the default values for all parameters. Thus, the only commands required are,

```
EMISSION EXPLOSIVE ;
EMIT EXPLOSIVE CATHODE ;
```

The following figure illustrates three stages of beam development in the Aurora diode.



Explosive emission from the Aurora diode.

EMISSION GYRO Command

Function: Specifies a gyro-emission process.

Syntax: EMISSION GYRO beam_current(t) magnetic_field p_longitudinal p_transverse guiding_center beam_center [options] ;

Arguments:

beam_current	-	total beam current (A), constant or defined in FUNCTION command.
magnetic_field	-	magnetic field at emitting surface (T).
p_longitudinal	-	longitudinal momentum (m/sec).
p_transverse	-	transverse momentum (m/sec).
guiding_center	-	distance from beam axis to guiding_center axis (m).
beam_center	-	point at which beam axis intercepts surface.
options	-	see EMISSION [options] command.

Defaults:

For this process, the creation_rate in 3D depends on the number of cells the annular emission zone crosses. Here, the creation_rate represents the number of particles created at each time step and uniformly distributed on a circle about the guiding_center. All other option defaults are the same. The EMISSION GYRO command must be entered for this model to be used.

Description:

The gyro-emission algorithm produces a beam of particles gyrating about a guiding_center axis parallel to the externally applied magnetic field. Particles are emitted from a circle on the surface, which must be conformal with a spatial coordinate. The guiding_center and beam axes must be normal to the surface. The radius of the circle will be determined from the specified magnetic_field and momentum components.

The gyro-emission arguments begin with the incident beam_current. You must also specify the magnetic_field on the emitting surface, and this should be consistent with the externally supplied magnetic field (COILS and PRESET, Ch. 19). The initial momentum is specified as two components of the relativistic momentum, gamma times velocity, in units of m/sec. The p_longitudinal component is parallel to the guiding_center axis and the p_transverse component is perpendicular to the axis.

The guiding_center specifies the distance between the beam axis and the guiding_center axis. Normally the guiding_center axis will be aligned with the coordinate system axis; in this case, guiding_center will be the radius at which emission is centered. The beam_center is the location of the centroid of the entire beam (usually not the guiding_center location). Note that only the beam_current is allowed to be a function of time.

In 3D simulations the number of particles in the beam annulus is roughly the number of cells around the annular emission zone. MAGIC determines the cell resolution on the emission surface and then creates the number of emission sites that will provide a smooth distribution of density around the annulus of emission. Here, the creation_rate represents the number of particles created at each time step and uniformly distributed on a circle about the guiding_center. All other option defaults are the same.

Restrictions:

1. Up to 20 EMISSION commands of all types may be used in a simulation.

2. In 2D simulations, gyro-emission is limited to cartesian (x,y) and cylindrical (z,r) coordinate systems.

3. In 3D simulations, gyro-emission is presently limited to the cartesian (x,y,z), cylindrical (z,r,ϕ) and polar (r,ϕ,z) coordinate systems. In addition, the guiding_center axis must be parallel to the z axis.

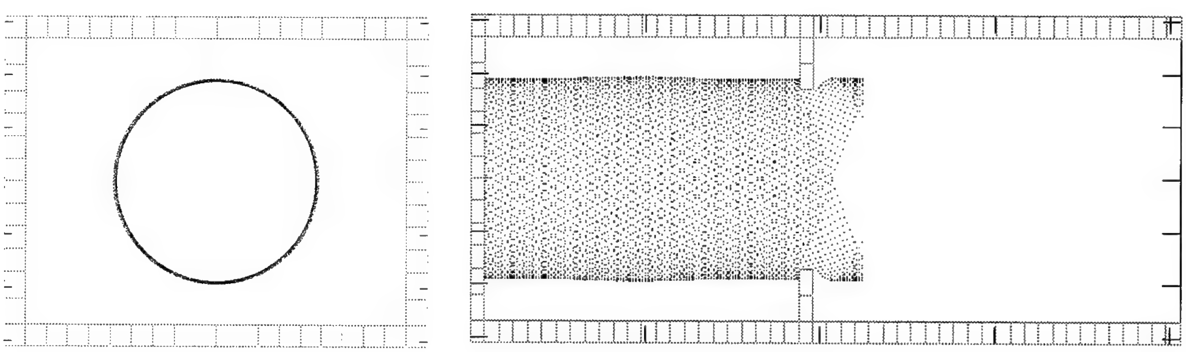
4. The magnetic_field specified should be consistent with the external guiding magnetic fields (BnST) that must be supplied using either the COILS and/or PRESET commands.

See Also: **FUNCTION, Ch. 6**
EMISSION [options], Ch. 16
EMIT, Ch. 16
COILS, Ch. 19
PRESET, Ch. 19

Examples:

The electron beam in rectangular gyrotron has a kinetic energy of 800 keV and beam current of 200 A. The guiding magnetic field is set to 0.15 tesla. The beam rotates at the gyro radius, and thus the guiding center radius is zero. The following figure illustrates electron phasespace in the xy and zy cross sections early in the simulation. The commands specifying the gyro emission parameters are:

```
Gcurrent = 200 AMPS ;  GuideRadius = 0.0 ;
Bguide = 0.15 TESLA ;
Parp = 5.E8 ; Perp = 5.E8 ;
POINT Center_Point 0.0, 0.0, 0.0 ;
EMISSION GYRO Gcurrent Bguide Parp Perp GuideRadius
   Center_Point
   MODEL GYRO.MODEL ;
EMIT GYRO.MODEL EMITTORPLATE ;
```



XY and ZY phase-space plot of gyrotron electrons.

EMISSION HIGH_FIELD Command

Function: Specifies a high-field emission process in a 2D simulation.

Syntax: EMISSION HIGH_FIELD a b phi(t,x1,x2,x3)
[options] ;

Arguments:

a	- Fowler-Nordheim constant, A (A/m).
b	- Fowler-Nordheim constant, B (m^{-1} $V^{-1/2}$).
phi	- work function (eV), constant or defined in FUNCTION command.
options	- see EMISSION [options] command.

Defaults:

None, except for the option defaults. The EMISSION HIGH_FIELD command must be entered for this model to be used.

Description:

The basic HIGH_FIELD emission process is described by the Fowler-Nordheim equation,

$$\frac{d^2q}{dAdt} = \frac{A\,E^2{}_s}{\phi\, t(y)^2} \exp\left(\frac{-\,Bv(y)\,\phi^{3/2}}{E_s}\right),$$

where A and B are the Fowler-Nordheim constants. The work function, ϕ, may be either a constant or a function with allowed dependence on time, t, and the spatial coordinates.

The remaining variables are computed internally. The normal electric field at the surface, E_s, is computed from the application of Gauss's law to the half-cell immediately above the emitting surface, or

$$E_s = (E_c A_c - q/\varepsilon_0) / A_s \ ,$$

where E_c is the electric field at the half-grid, A_c and A_s are the cell areas at half-grid and surface, respectively, and q is the existing charge in the half-cell.

The functions t(y) and v(y) are approximated by

$$t(y)^2 = 1.1$$

$$v(y) = 0.95 - y^2$$

$$y = 3.79\text{x}10^{-5}\, E_s^{1/2} / \phi,$$

where y is the Schottky lowering of the work function barrier.

Restrictions:

1. Up to 20 EMISSION commands of all types may be used in a simulation.

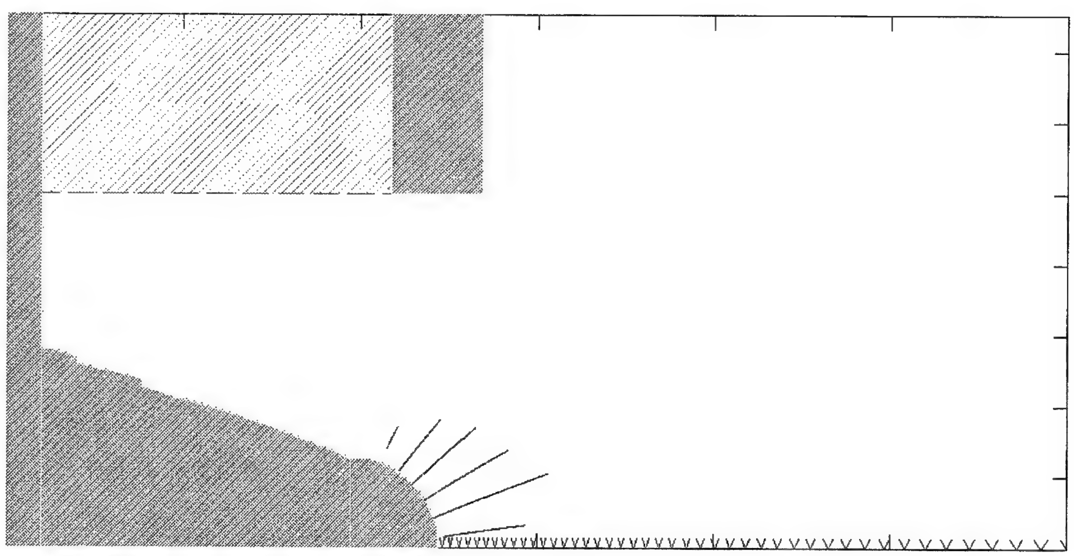
2. This emission process is available only in 2D simulations.

See Also: **FUNCTION, Ch. 6**
EMISSION [options], Ch. 16
EMIT, Ch. 16

Examples:

This 2D simulation example illustrates HIGH_FIELD (Fowler-Nordheim) emission from a field emitter tip. The geometry is a single field emitter tip shown in the following figure. The tip radius was assumed to be 25 microns. The gate and anode voltages were taken to be 200 and 300 volts, respectively. The emission model for this example creates one particle per cell on every fifth time step. The work function was assumed be five eV. This model, named FEA, is specified using the following commands:

```
A = 1.5414E-6 ;
B = 6.8308E+9 ;
PHI = 5.0 ;
FUNCTION NCREAT(T) = STEP(T,TIMECONST1) ;
EMISSION HIGH_FIELD A B PHI SURFACE_SPACING UNIFORM
     MODEL FEA TIMING 5 NUMBER NCREAT ;
EMIT FEA TIP;
```



Trajectories of a Field Emitter tip. The fine grid results in a femtosecond time step; consequently, tens of thousands of electromagnetic time steps are required for the particles to cross from the tip of the anode.

EMISSION PHOTOELECTRIC Command

Function: Specifies a photoelectric emission process.

Syntax: EMISSION PHOTOELECTRIC photon_source f(e) yield photon_energy
[options] ;

Arguments:

photon_source - name of photon source, defined in PHOTON command.
f(e) - electron energy function (electron/kev), defined in FUNCTION command (type DATA only).
yield - conversion yield (electrons/photon).
photon_energy - average photon energy (keV).
options - see EMISSION [options] command.

Defaults:

None, except for the option defaults. The EMISSION PHOTOELECTRIC command must be entered for this model to be used.

Description:

The photoelectric emission process is described by the equations,

$$\frac{d^5q}{dAdtdE\sin\theta\, d\theta} = \frac{1}{\pi}\,\eta\, s(t)\, f(E)\cos\theta$$

where

$$\eta = -41.87\eta'\chi(r)/\varepsilon \ .$$

Here, E represents the electron energy (keV), θ is the polar angle with respect to the surface normal, η is the conversion yield (electrons/photon), χ(r) is the fluence (cal/cm^2), ε is the average photon energy (keV), s(t) is the photon pulse time history (sec^{-1}), and f(E) is the electron energy distribution (electrons/keV).

Both the photon time history, s(t), and the fluence, χ(r), must be specified in a PHOTON command. The photon source model must reference the name used in the PHOTON command. The electron energy function, f(E), must be specified using type DATA in a FUNCTION command. The data will consist of pairs representing groups of electron energy (keV) vs. relative group normalization. Only relative normalization is required for the energy distribution; the correct normalization will be automatically supplied. The angular distribution, sinθ cosθ, is built into the algorithm using a ten-group, equal probability representation. The remaining arguments are the conversion yield (electrons/photon) and the average photon_energy (kev), representing η and ε respectively.

Restrictions:

1. Up to 20 EMISSION commands of all types may be used in a simulation.
2. The EMISSION PHOTOELECTRIC command can only be used with the ELECTRON species.

See Also: FUNCTION, Ch. 6
EMISSION [options], Ch. 16
EMIT, Ch. 16
PHOTON, Ch. 16

References:

A. McIlwain and B. Goplen, "Representation of Photoemission from Blackbody Spectra," Science Applications, Inc. Report, SAI-75-516-AQ, January 1976.

B. Goplen, J. Brandenburg, and R. Worl, "Canonical SGEMP Simulation Problems," Mission Research Corporation Report, MRC/WDC-R-109, also, AFWL-TN-86-57, March 1986.

Examples:

We consider emission from silicon caused by a symmetric, triangular, 10-nsec FWHM pulse of five-keV blackbody originating at coordinates (0,0) and propagating as a plane wave in the positive x direction. The yield for silicon is 10^{-3} electrons/photon, and the average photon energy is 13.6 keV. The (crude) electron energy distribution consists of five equally probable groups with midpoint energies of 1, 2, 4, 7, and 10 keV.

The model specifies three particles per cell and a creation frequency that matches the electromagnetic time step. Creation is fixed at a distance of 10^{-5} m from the surface but is random transversely. This model, named STD, is specified in the following commands:

```
! Define photon time history
FUNCTION S DATA 3  0.0,0.0  1.0E-8,1.0E+8  2.0E-8,0.0 ;
!
! Define electron energy distribution
FUNCTION F DATA 5  1.0,0.2  2.0,0.2  4.0,0.2  7.0,0.2  10.0,0.2 ;
!
! Define photon source
PHOTON SOURCE 1.0 S 0.0 PLANE-X1 0.0 0.0 ;
!
! Specify the photoelectric emission model
CONV = 1.0E-3 ;
EAVE = 13.6 ;
EMISSION  PHOTOELECTRIC SOURCE F CONV EAVE OUTWARD_SPACING RANDOM 1E-5;
!
! Emit from the spatial object labeled silicon
EMIT PHOTOELECTRIC SILICON ;
```

EMISSION SECONDARY Command

Function: Specifies a secondary electron emission process in a 3D simulation.

Syntax:

```
EMISSION SECONDARY peak_delta energy_at_peak
        [ SPECIES primary secondary ]
        [ options ] ;
```

Arguments:

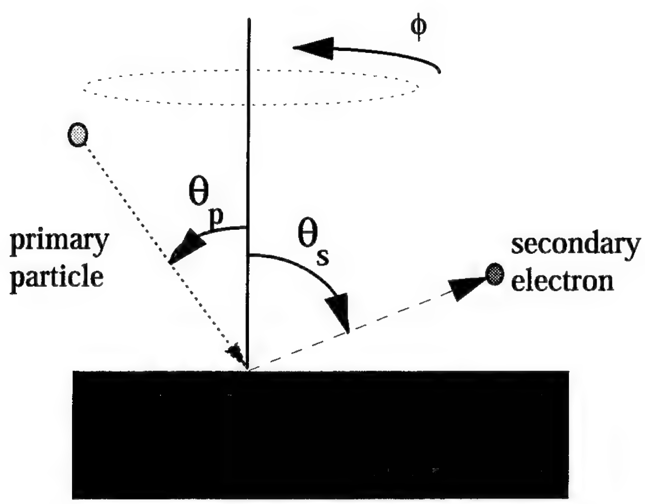
peak_delta - maximum secondary emission coefficient at normal incidence (unitless).
energy_at_peak - primary particle energy at peak_delta (eV).
primary - primary particle species (ELECTRON, PROTON, or defined in SPECIES command).
secondary - secondary particles species (ELECTRON, or defined in SPECIES command).
options - see EMISSION [options] command.

Defaults:

The default primary species is ELECTRON, and the default secondary species is ELECTRON. The EMISSION SECONDARY command must be entered for this model to be used.

Description:

This command enables secondary electrons to be emitted from the surface of a conducting spatial object when it is impacted by a primary particle. The following figure illustrates the coordinate system. The polar angle of the primary particle, θ_p, and the secondary electron, θ_s, are measured with respect to the surface normal.



Coordinate system illustrating primary particle and secondary electron.

In our model, the secondary emission coefficient dependence upon primary particle energy is given by a standard form and the material-dependent parameters, peak_delta and energy_at_peak. The complete equation governing secondary emission is

$$\frac{dq_s}{dq_p dE_s d\theta_s \sin\theta_s d\vartheta_s} = \frac{1}{2\pi}\delta(E_p,\theta_p) f(E_s / E_p)$$

where the secondary emission coefficient, δ, is a product of two functions, one in primary polar angle and one in primary energy,

$$\delta(E_p,\theta_p) = \frac{\delta_0(E_p)}{\cos(\theta_p)}$$

$$\delta_0(E_p) = \delta_{peak}((2.72)^2 \frac{E_p}{E_{peak}} \exp\left[-2\left(\frac{E_p}{E_{peak}}\right)^{1/2}\right]$$

Here, δ_0 is the secondary emission coefficient at normal incidence for the primary. The variables, δ_{peak} and E_{peak}, are represented by the arguments, peak_delta and energy_at_peak, respectively. An artificial cutoff in the polar angle dependence (at 85 degrees) is employed to prevent the coefficient from going to infinity for grazing angles.

For the emitted secondary electron, the energy and spherical angle distributions are normalized to unity.

$$\int f(E_s / E_i) dE_s = 1$$

$$\int \sin(\theta) d\theta_s d\vartheta_s = 1$$

The secondary energy distribution is peaked at 7.5 eV with a full-width at half-maximum of 10 eV. The angular distribution is evidently homogeneous, having no preferred direction.

Since the default primary and secondary particles are both ELECTRON, cascading can result; i.e., the secondaries can immediately generate more secondaries. To prevent this, you can use the SPECIES option to re-label the secondary particle species. For example, primary ELECTRON particles can be used to generate SECONDARY particles only. Thus, physical cascading is easily prevented, but important physical effects may be lost. The most important statistical parameter is the creation_rate (EMISSION [options], Ch. 16), which controls the number of macroparticles (as opposed to the amount of charge). The combination of physical cascading with a creation_rate greater than unity can cause numerical cascading (too many numerical particles). To prevent it, you can reduce the creation_rate from its default value to unity.

Restrictions:

1. This emission process is available only in 3D simulations.

2. Up to 20 EMISSION commands of all types may be used in a simulation.

See Also: **EMISSION [options], Ch. 16**
EMIT, Ch. 16
SPECIES, Ch. 18

Examples:

This example illustrates secondary electron emission from a copper conductor named "Stage1" caused by impact of an electron beam. For copper[1], the emitted number coefficient has a peak_delta of 1.3 at a primary energy_at_peak of 500 eV. To control any tendency to cascade artificially, we define a new electron species named "BLUE" and enable emission only from the ELECTRON species (as primary). Each incident macroparticle creates the default number of one secondary particle.

```
SPECIES SECONDARY CHARGE 1 MASS 1 ELECTRON ;
EMISSION SECONDARY 1.3 500
     SPECIES ELECTRON SECONDARY
     NUMBER 1;
EMIT Stage1 SECONDARY ;
```

[1] E. Sternglass, Westinghouse Res. Sci. Paper 1772 (1954).

EMISSION THERMIONIC Command

Function: Specifies a thermionic emission process in a 2D simulation.

Syntax: EMISSION THERMIONIC work_function(t,x1,x2,x3) temperature(t,x1,x2,x3)
[options] ;

Arguments:
work_function - work function (eV), constant or defined in FUNCTION command.
temperature - temperature (deg K), constant or defined in FUNCTION command.
options - see EMISSION [options] command.

Defaults:

None, except for the option defaults. The EMISSION THERMIONIC command must be entered for this model to be used.

Description:

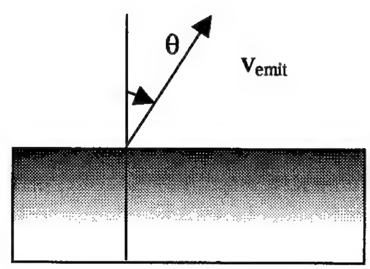
The thermal emission process is described by the Richardson-Dushman equation,

$$\frac{d^2 q}{dAdt} = A_0 T^2 \exp\left(\frac{-\phi_w}{kT}\right),$$

where k is the Boltzmann constant and A_0 is the Dushman parameter (1.204×10^6 A/m^2 Kelvin^2). The work function, ϕ_w, may be either a constant or a function with allowed dependence on time, t, and the spatial coordinates. The same is true of the temperature. The velocity distribution of emitted particles is Maxwellian in energy and sin-of-polar angle from vertical, which pre-produces uniform directionality in 3D:

$$\frac{d^6 n}{d^3 v d^3 x} = \exp\left(\frac{-mv^2}{2kT}\right), \text{ and}$$

$$\frac{d^6 j_{emit}}{d\,v d\phi\, d\vartheta\, d^3 x} = v^3 \sin(\vartheta)\cos(\vartheta)\exp\left(\frac{-mv^2}{2kT}\right).$$



Restrictions:

Up to 20 EMISSION commands of all types may be used in a simulation.

See Also:
FUNCTION, Ch. 6
EMISSION [options], Ch. 16
EMIT, Ch. 16

EMIT Command

Function: Enables emission from the surface of a conducting spatial object.

Syntax: EMIT model { area, volume } [{ EXCLUDE, INCLUDE } { area, volume }] ... ;

Arguments:

model	- name of emission model, defined in EMISSION command.
area	- name of conformal area in 2D simulation, defined in AREA command.
volume	- name of conformal volume in 3D simulation, defined in VOLUME command.

Description:

The EMIT command enables particle emission everywhere on the surface of a spatial object. In 2D simulations, the object can only be an area; in 3D simulations, the object can only be a volume. In addition, the spatial object must be assigned the property of being perfectly conducting (CONDUCTOR, Ch. 14). Only two arguments are required. The first is the model name specified in an EMISSION command, and the second is the name of the spatial object. Note that emission will be enabled over the entire surface of the specified object.

If emission is to be restricted from certain spatial regions on the object, these may be specified using the EXCLUDE / INCLUDE options. EXCLUDE restricts emission from a specified portion of the conducting object. INCLUDE reverses the effect of EXCLUDE. These options may be applied in any number and order to tailor the emission surface on the spatial object. The objects specified using the EXCLUDE / INCLUDE options must be conformal, but need not be assigned any material property; they may be defined arbitrarily for convenience. However, they must intersect some portion of the conducting object to have any effect.

Restrictions:

1. Up to thirty EMIT commands may be used in a simulation.

2. Only AREA spatial objects can emit in 2D simulations and VOLUME objects in 3D simulations.

3. The emitting object can have any shape, but must be specified as perfectly conducting in a CONDUCTOR command.

4. Non-emitting regions must be designated with EXCLUDE / INCLUDE options. These objects need not be conducting, but must be conformal in shape.

See Also:
AREA, Ch. 10
VOLUME, Ch. 10
CONDUCTOR, Ch. 14
EMISSION [options], Ch. 16

Examples:

In this example, we enable beam-emission model STD on a spatial object named "Cathode," while excluding it from the "Top" and "Bottom" of Cathode. The command is

```
EMIT STD Cathode EXCLUDE Top EXCLUDE Bottom ;
```

Equivalently, we could have defined a separate object named "Middle" and simply enabled emission from it.

PHOTON Command

Function: Specifies photon source in time and space.

Syntax: PHOTON photon_source chi(r) s(t) advance point ;

Arguments:

photon_source	-	name of photon source, user-defined.
chi(r)	-	spatial function, user-defined in FUNCTION command (cal/ sq. cm).
s(t)	-	temporal function, user-defined in FUNCTION command (1/sec).
advance	-	retarded time advancement (m).
point	-	spatial coordinates, or name of point defined in a POINT command.

Description:

The PHOTON command allows the user to specify one or more photon sources to drive photoelectric emission processes. Each model so specified must be given a unique name. The photon_source name will be referred to by the EMISSION PHOTOELECTRIC command, which specifies the photoelectric emission model.

The photoelectric-emission process is described by the equation,

$$\frac{d^5 q}{dAdtdE \sin\theta \, d\theta} = \frac{1}{\pi} \eta s(t) f(E) \cos\theta,$$

where

$$\eta = -41.87(\eta' / \xi)\chi(r).$$

The PHOTON command specifies the spatial function, $\chi(r)$, and the temporal function, s(t). The time used in the temporal function is retarded, i.e., reduced by the spatial distance (divided by the speed of light). For both functions, the spatial distance is computed from the photon origin, specified by the spatial point, to a point on the emitting spatial object (EMIT, Ch. 16). Thus, the character of the wave front (spreading) is contained entirely in the spatial function, $\chi(r)$. This is sufficient to model simple sources, i.e., plane waves, etc. To model extended but finite photon sources, multiple PHOTON commands could be used. The retarded time advancement should be used initially to position the wave front near the emitting structure.

Restrictions:

1. The number of PHOTON commands in a simulation is limited to five.

2. The temporal function must be normalized to unity.

See Also: **FUNCTION, Ch. 6**
EMISSION PHOTOELECTRIC, Ch. 16

Examples:

Consider photoelectric emission caused by a symmetric, triangular, photon pulse of 10 nsec half width, originating at coordinate (0, 0), and propagating as a plane wave in the positive x_1 direction. (This photon source model is arbitrarily given the name, Source.) The conversion yield is 10^{-3} electrons/photon and the average

photon energy is 13.6 keV. The resulting electron energy function consists of five equally probable groups at energies of 1, 2, 4, 7, and 10 keV. (This electron energy function is arbitrarily given the name, EFN.)

Particle creation is fixed on, and random over, the emitting cell surface. There are three particles per cell created on every time step. This model is specified in the commands,

```
FUNCTION EFN DATA 5 1.0 0.2 2.0 0.2 4.0 0.2 7.0 0.2 10.0 0.2;
EMISSION PHOTOELECTRIC Source EFN 1.0E-3 13.6
     SURFACE_SPACING RANDOM OUTWARD_SPACING RANDOM 0.0 ;
FUNCTION Temporal DATA 3 0.0 0.0 1.0E-8 1.0E8 2.0E-8 0.0 ; .
POINT Source_Origin 0., 0. ;
PHOTON Source 1. Temporal 0.0 Source_Origin ;
```

IONIZATION Command

Function: Specifies a neutral-gas ionization model for particle creation in a 2D simulation.

Syntax:

```
IONIZATION neutralgas pressure temperature species1 species2
        { SOURCE rate(t,x1,x2,x3), IMPACT species_i }
        [THERMAL energy]
        [{EXCLUDE, INCLUDE} area]
        [START_TIME starttime]
        [END_TIME endtime]
        [TIMING step_multiple]
        [NUMBER creation_rate(t,x1,x2,x3)]
        [LOSS, NOLOSS]
        [SPACING {CENTERED,RANDOM}] ;
```

Arguments:

neutralgas	- name of neutral gas as defined in the MATERIAL table.
pressure	- neutral gas pressure in pascals.
temperature	- neutral gap temperature in °K.
species1,2	- species comprising the ionization pair, = ELECTRON, = name of ion species defined in SPECIES command, or = INACTIVE.
rate	- production rate of charged species (number/m^3/s), constant or defined in FUNCTION command.
species_i	- incident species (typically ELECTRONs) for impact ionization,
energy	- initial thermal energy of ions (eV), constant.
starttime	- time in seconds at which ionization starts.
area	- name of spatial area, defined in AREA command.
endtime	- time in seconds at which ionization ends.
step_multiple	- LORENTZ time step multiplier (integer).
creation_rate	- particle creation (particles/cell/time step), constant or defined in FUNCTION command.

Description:

The IONIZATION command controls the generation of charged particles due to ionization of a background neutral gas. The user must specify the two species, species1 and species2, comprising the ionization pair, typically ELECTRON and the name of some positive ion created with the SPECIES command.

The argument, neutralgas, is the name of the background gas. Its properties, must be defined in the MATERIALs tables or by using the MATERIAL command. The required properties are ATOMIC_NUMBER, ATOMIC_MASS, and CROSS_SECTION coefficients. (For polyatomic gases, specify the molecular number and mass values.) In addition, you must specify the gas pressure and temperature. The number density is calculated from the ideal gas law. Finally, you may select either the SOURCE term ionization model or the IMPACT ionization model for the creation of electron-ion pairs. Each model represents different physical processes. Regardless of which ionization model is selected, the background gas causes an energy loss to kinetic particles due to impact. Bethe's energy loss equation is used to degrade the momentum of ambient electrons. The equation for high energies is

$$-dE/dx = 2\pi\, mc^2\, r_e^2\, N\, Z_{gas} / \beta^2 \{ \ln[\beta^2\gamma^2(\gamma-1)/(2I_e^2)] - 1/\gamma(2-1/\gamma)\ln 2 + 1/\gamma^2 + 1/8(1-1/\gamma)^2 \},$$

where r_e is the classical electron radius, N is the atomic (or molecular) number density of the gas, and Z_{gas} is the atomic (or molecular) electron charge of the gas. I_e is the average effective ionization potential in units of mc^2. It is parameterized by

$$I_{eff} = 9.1\ Z\ (1+1.9Z^{-2/3})\ \text{(in eV)}.$$

The SOURCE model is the most direct, with essentially no built-in physics; the user simply supplies the ionization rate directly. The SOURCE model requires a user-supplied rate function, which may vary in time and space:

$$dn^-/dt = dn^+/dt = \text{rate}(t,x_1,x_2,x_3).$$

where: r = ionization rate function
n^- = negative particle species number density
n^+ = positive particle species number density.

The IMPACT model determines the ionization rate based on the cross section coefficients associated with the particular target neutral. The energy-dependent cross section is parameterized by the following equation:

$$\sigma_{ion} = 0.5\ [\ h^2\ /\ (mc\beta)^2\ ]\ [\ M^2\ (\ 2\ \ln(\gamma\beta) - \beta^2) + C]$$

where $0.5\ [\ h^2\ /\ (mc)^2\ ] = 1.874 \times 10^{-20}\ cm^2$, γ is the relativistic factor, β ratio of the velocity to the speed of light, and M^2 and C are the cross-section coefficients defined in the MATERIAL table. The rate equation for the impact model is given by the following equation:

$$dn^-/dt = dn^+/dt = \sigma_{ion}\ n_t\ \phi_I\ ,$$

where: σ_{ion} = total cross section for impact ionization,
n_t = target neutrals number density, and
ϕ_i = incident particle flux, e.g., $n_i|v_i|$.

When ionization particles are created, they are given a momentum based on a Maxwellian distribution which assumes a thermal energy equal to kT, where T is the gas temperature. The THERMAL option allows you to specify the ionization kinetic energy distribution.

The ionization model can add significantly to the run-time of a simulation. It is, therefore, cost-effective to turn-off the ionization algorithm completely wherever and whenever the source function is known a priori to be zero, for example, in regions of space such as dielectrics, where ionization is suppressed, or before and after an ionizing laser pulse has traveled through the simulation. EXCLUDE / INCLUDE options allow the active spatial region of the ionization model to be restricted, and START_TIME and END_TIME options allow the active time period of the ionization model to be restricted. INCLUDE reverses the effect of EXCLUDE, and these two options may be applied in any number and order. The EXCLUDE / INCLUDE, START_TIME, and STOP_TIME options will override the ionization source rate and suppress ionization, even if the rate is nonzero.

For short time-scale simulations in which ion motion is negligible, it may be desirable to follow just the electrons; this is accomplished by using the word INACTIVE for one of the species. You must specify a constant

background neutral gas density which is assumed to exist within the simulation volume at the beginning of the simulation. During the simulation, this gas is depleted by ionization events, as specified by SOURCE, IMPACT, or AVALANCHE models. If the neutral gas should become fully ionized, then ionization, e.g., particle creation, turns off.

By default, the ionization model creates particles every LORENTZ (not MAXWELL) time step. If this results in an excessive number of particles, then the user may wish to use the TIMING option to force particle creation to occur only on an integer step_multiple of the LORENTZ time step.

For SOURCE creation of ionization particles, the ionized charge is placed at the center of a cell. An option exists to improve the statistical control by increasing the NUMBER of particles created each time step. The increased creation_rate may be a constant or a function of both time and space. The SPACING option allows for either the default cell-CENTERED spacing or RANDOM spacing within the cell.

See Also: **MATERIAL, Ch. 14**
SPECIES, Ch. 18
POISSON, Ch. 19
POPULATE, Ch. 19
PRESET, Ch. 19

Restrictions:

1. The IONIZATION command can be used only in 2D simulations.

2. The maximum number of IONIZATION commands is five.

Examples:

The following commands enable the ionization model to create ionized nitrogen in a room temperature gas having a pressure of 10.7 mtorr. In this example, ionization sweeps across the confinement chamber at the speed of light. The pulse width of the ionization source is 1/10th of the axial dimension of the chamber. A region call, DIEL, corresponding to a dielectric window is excluded from the gas region. The creation pair is centered in each cell.

```
REAL NEUTRAL_DENSITY,ION_ENERGY ;
NEUTRAL_DENSITY = 10E13 ;
ION_ENERGY = 25 ;
DELTA_TIME = SYS$DTIME ;
QUESS_RATE = 0.010*NEUTRAL_DENSITY/DELTA_TIME ;
PULSE_WIDTH = (SYS$X1MX-SYS$X1MN)/10 ;
XSTART = SYS$X1MN ;
FUNCTION FIXEDRATE(T,X1,X2) = QUESS_RATE ;
FUNCTION SCANRATE(T,X1,X2) =
    QUESS_RATE*STEP(XSTART+1.C*T,X1)*STEP(X1,XSTART+1.C*T-PULSE_WIDTH);
TSTART = 0 ;
TEND   = TSTART+(PULSE_WIDTH+SYS$X1MX-SYS$X1MN)/1.C ;
KCREATION_RATE = 1  ;
NUMBER_PER_CELL = 1 ;
 PRESSURE = 10.7MILLITORR ;
```

```
 TEMPERATURE = 300 ;
IONIZATION NITROGEN pressure temperature NITROGEN ELECTRON
      SOURCE SCANRATE
      THERMAL ION_ENERGY
       EXCLUDE DIEL
       START_TIME TSTART
       END_TIME   TEND
       TIMING KCREATION_RATE
       NUMBER NUMBER_PER_CELL
       SPACING CENTER ;
```

17. ELECTROMAGNETIC FIELDS

This Chapter covers the following commands:

TIME_STEP
MODE (2D simulations only)
MAXWELL CENTERED
MAXWELL QUASI_NEUTRAL
MAXWELL QUASI_STATIC
MAXWELL HIGH_Q
MAXWELL BIASED

The MAXWELL algorithms solve Maxwell's equations to obtain electromagnetic fields (E1, E2, E3, B1, B2, B3), which vary in time and space. (Other algorithms calculate electrostatic fields and eigenfunctions (POISSON and EIGENMODE, Ch. 19), but only the MAXWELL algorithms calculate time-varying fields.) You can use the MAXWELL commands to select an electromagnetic algorithm and/or parameters and the TIME_STEP command to set the time step. In 2D simulations, you can use the MODE command to select either transverse magnetic mode fields (E1, E2, B3) or transverse electric mode fields (E3, B1, B2). (The MODE command does not apply to 3D simulations, which always include all six field components.)

You may not need to use any of these commands. The default electromagnetic solution uses the following parameters:

time step	- 80% of centered-difference Courant criterion
mode	- BOTH (both TE and TM) modes, for 2D simulations only
algorithm	- CENTER (time-centered)

This combination provides a very conservative, robust solution for use with non-relativistic particles. It is consistent with the particle algorithm defaults (Ch. 18). If you are unsure of your simulation requirements, this is the recommended configuration.

On the other hand, you may elect to use the other electromagnetic algorithms to speed up the simulation, to improve fidelity by matching the algorithm to the physics, or even to suppress certain physical effects to gain insight. Algorithm selection is especially important, and the following table indicates conditions that might favor one algorithm over another. For example, if particles are either absent or non-relativistic, then the CENTERED algorithm generally provides superior speed.

General conditions for algorithm selection.

MAXWELL ALGORITHM	CENTERED	HIGH_Q	BIASED	QUASI_STATIC	QUASI_NEUTRAL
Speed required	X				
Long time scales				X	
No particles	X				
Very slow particles				X	
Slow particles	X				
Relativistic particles		X	X		
Cavities and particles		X			
High particle noise			X		

A brief explanation of the principal factors that impact the electromagnetic field solution follows.

1. Field definition — The electromagnetic fields (E1, E2, E3, B1, B2, B3) are defined at discrete locations in space and time. The following figure illustrates the spatial definition in the unit cell. The electrostatic fields (E1ST, E2ST, E3ST) and magnetostatic fields (B1ST, B2ST, B3ST) are coincident with their dynamic counterparts. The current densities (J1, J2, J3) are spatially coincident with the electric fields, and the charge density (Q0) is at the corner of the cell (intersection of the electric field vectors). The fields are also defined at discrete values of time (TIME_STEP, Ch. 17), with the electric and magnetic fields typically separated by half a time step.

2. Cell shape - The 3D cell shown is cubical; however, in practice it will be elongated (GRID, Ch. 11), and the sides will be curved (or non-parallel) in polar and spherical coordinates (SYSTEM, Ch.10). The 2D cell is obtained simply by viewing the 3D cell in the direction of the ignorable coordinate (x_3). The fields are the same at all values of x_3, which is the coordinate of perfect symmetry in all 2D simulations.

3. Spatial grid — Accuracy in the electromagnetic solution is determined primarily by spatial resolution, i.e., the cell size (GRID and AUTOGRID, Ch. 11) relative to the wavelength. The classic example is that calculated eigenvalues are always downshifted from their physical values, due simply to the discrete representation of fields in space. In addition, there is an upper limit to the frequency that can be supported by the grid. Any wavelength shorter than about six cells will degenerate into noise and be lost from the solution.

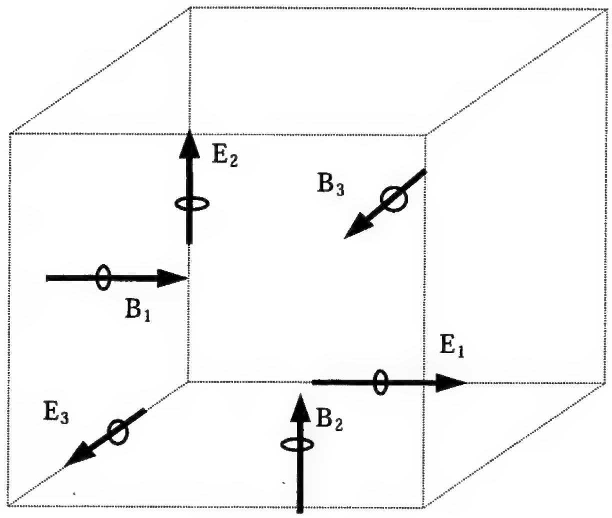
4. Time step — All MAXWELL algorithms use a time step to advance the fields in time. The size of the time step has virtually no effect on accuracy, so it is generally advisable to use a large time step to minimize expense. However, a Courant stability criterion limits the size of the time step, and each algorithm has a different limit. Rapid, catastrophic failure results from exceeding this limit. Also, outer boundaries (PORT, Ch. 12) and particle dynamics (LORENTZ, Ch. 18) may effectively limit the size of the electromagnetic time step.

5. Damping — Two of the MAXWELL algorithms (HIGH_Q and BIASED) contain damping designed to reduce high-frequency fields (particularly noise from relativistic particles). However, some damping occurs at all frequencies, including those of physical interest. In other words, each algorithm has an intrinsic, frequency-dependent Q. If the algorithm damping exceeds the actual physical damping, incorrect fields may result with adverse effects on results for saturation, energy balance, efficiency, etc.

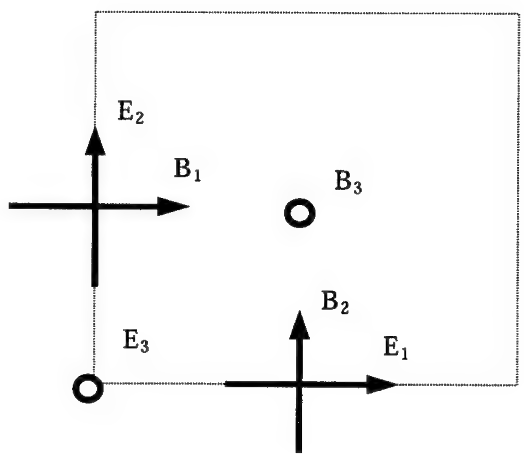
6. TE and TM modes — In 2D simulations, both the transverse electric (TE) mode and the transverse magnetic (TM) mode are calculated by default. The TE fields (B1, B2, E3) and the TM fields (E1, E2, B3) may coexist independently, or they may be coupled through particle dynamics and/or unique structures (POLARIZER, Ch. 15). For 2D simulations which require only the TM mode (a common occurrence), eliminating the TE mode will double the speed. (Pure TE mode simulations, which omit all space-charge effects, are also possible but are performed only rarely.) In 3D simulations, all six fields (E1, E2, E3, B1, B2, B3) are always computed, even though individual components may vanish under certain conditions.

7. Particles — All the MAXWELL algorithms may be used with particles (SPECIES, Ch. 18) to produce self-consistent simulations that fully account for field-particle interactions. All account for space charge exactly, i.e., they satisfy Gauss's law at the cell level, given that the continuity equation is conserving (CONTINUITY, Ch. 18). The MAXWELL algorithms differ in that they accommodate different velocity regimes, ranging from very low to highly relativistic. Plasma simulation can be especially difficult when the Debye length is unresolved spatially or the plasma frequency is unresolved temporally.

8. Materials, etc. — All MAXWELL algorithms are compatible with the full range of outer boundaries (Ch. 12) and material properties (Ch. 14 and 15). However, the number of possible combinations is infinite, and diligence (i.e., testing) is always advised.



(a) 3D simulation



(b) 2D simulation

Spatial definition of fields.

TIME_STEP Command

Function: Specifies the electromagnetic time step.

Syntax: TIME_STEP time_step ;

Arguments: time_step - electromagnetic algorithm time step (sec).

Defaults:

If you do not specify the time_step, a default value equal to 80% of the centered-difference Courant criterion will be used.

Description:

You can use the TIME_STEP command to specify the time_step used in the electromagnetic algorithm. If you do not specify the time_step, a default value equal to exactly 80% of the centered-difference Courant criterion will be used, irrespective of the choice of algorithm.

The centered-difference Courant ratio squared stability criterion is given by $\chi < 1$, where

$$\chi^2 = c^2 \delta t^2 \sum_{i=1}^{N} \frac{1}{(\delta x_i)^2}.$$

is the Courant ratio squared, δx_i is the cell size in meters, and N is the number of dimensions (2 or 3). Once the spatial grid is completed, it is automatically searched to find the most restrictive cell. Then the default time_step is calculated using $\chi = 0.8$. This time_step is also entered into the system variable, SYS$DTIME. This system variable is accessible in the input and can then be used to alter the time_step (see Examples, below).

The centered-difference criterion is used as the default because it is safe. It is conservative for all electromagnetic algorithms and coordinate systems. It is safe with all outer boundaries that use a time step. It is also safe when used in the particle kinematics. However, there may be circumstances in which you want to alter the time step. For example, you may wish to take advantage of the larger time_step possible with other algorithms, or you may wish to make a minor adjustment of the time_step for periodicity (see Examples, below). In some cases, it may be necessary to reduce the time_step significantly to accommodate particle resolution requirements (LORENTZ, Ch. 18). In 2D or 3D simulations that have symmetry such that they are perfectly one-dimensional, the Courant criterion can be violated with impunity, and a very large multiple can be chosen.

Restriction:

The centered-difference criterion is used to calculate the default time_step, irrespective of the algorithm actually used.

See Also:
AUTOGRID, Ch. 11
GRID, Ch. 11
MAXWELL algorithms, Ch. 17
CONTINUITY, Ch. 18
LORENTZ, Ch. 18

Example:

Suppose that the magnitude of the default time_step is satisfactory, but that you want to adjust it very slightly so that an exact (integer) number of time_steps equals some specified period. (This is useful in certain cavity-tuning procedures). You can achieve the desired effect with the following commands, using 10 GHz as an example.

```
frequency = 10 GHz ;
period = 1.0 / frequency ;
I = period / SYS$DTIME ;
dt = period / I ;
TIME_STEP dt ;
```

Note that dt differs only slightly from the default time_step.

MODE Command

Function: Specifies the electromagnetic mode in a 2D simulation.

Syntax: MODE { TE, TM, BOTH } ;

Arguments: none.

Defaults:

The default mode in a 2D simulation is BOTH. You can use the MODE command to select TE or TM, thus suppressing the other mode.

Description:

You can use the MODE command to select the electromagnetic mode in a 2D simulation. The modes are transverse electric (TE) with fields B1, B2, and E3 and transverse magnetic (TM) with fields E1, E2, and B3. (Here, TE and TM simply identify which electromagnetic fields will be present. These should not be confused with "waveguide modes." Also, the transverse electromagnetic (TEM) mode is not a separate mode, but is actually a degenerate case of the TM mode.)

The majority of 2D simulations produce only a transverse magnetic (TM) mode. Certain rare cases produce only a transverse electric (TE) mode. If both TE and TM modes are produced, they will co-exist independently unless they are coupled through 3D particle kinematics and/or unique structures such as the polarizer (POLARIZER, Ch. 15).

If you can suppress one mode, the electromagnetic solution speed will double. To ascertain whether both modes are required, you must consider the interaction between the field and current density components in the Maxwell and Lorentz equations. If you are unsure, use the default and inspect the results. If all the fields in either mode are zero, then that mode is not required. Even if a mode exists, you may decide to suppress it for speed if it is not relevant to the dynamics. Also, you may wish to suppress a mode to study particular physical effects. For example, suppressing the TM mode eliminates space charge.

Restrictions:

The MODE command can be used only in 2D simulations.

See Also:
MAXWELL algorithms, Ch. 17
CONTINUITY, Ch. 18
LORENTZ, Ch. 18

Examples:

You can suppress the TE mode in a 2D simulation with the following command.

```
MODE TM ;
```

MAXWELL CENTERED Command

Function: Specifies centered-difference electromagnetic algorithm.

Syntax: MAXWELL CENTERED ;

Arguments: none.

Defaults:

Defaults are set for all electromagnetic field algorithm parameters. The default algorithm is CENTERED. You can change this with one of the MAXWELL commands. You can use the TIME_STEP command to specify the time_step. If you do not specify the time_step, a default value equal to 80% of the centered-difference Courant criterion will be used, irrespective of the MAXWELL algorithm selected.

Description:

The MAXWELL CENTERED command specifies a centered-difference solution of the fully time-dependent Maxwell's equations. This algorithm is the simplest of the time-dependent field algorithms. It is suitable for use without particles or with non-relativistic particles.

The centered-difference Courant criterion is given by $\chi < 1$, where

$$\chi^2 = c^2 \delta t^2 \sum_{i=1}^{N} \frac{1}{(\delta x_i)^2}.$$

is the Courant ratio squared, δx_i is the cell size in meters, and N is the number of dimensions (2 or 3). In certain cases (SYMMETRY, Ch. 12), the criterion may be even more restrictive; however, $\chi < 0.80$ is generally safe. Despite the relatively restrictive time step, the centered-difference algorithm is the fastest in terms of covering a given time span.

Another characteristic of the centered-difference algorithm is that it provides no damping at any frequency (the algorithm Q is infinite). Thus, while excellent for purely electromagnetic simulations, center-difference is very susceptible to the high-frequency noise typically produced by relativistic particles. In extreme form, this susceptibility appears as field aliasing (alternating signs in the cell fields). Therefore, we recommend use of the centered-difference algorithm only for purely electromagnetic simulations or with non-relativistic particles.

Restrictions:

1. This algorithm has an upper limit to the time step used.

2. It should be used only in particle-free or non-relativistic simulations.

See Also:
SYMMETRY AXIAL, Ch. 12
TIME_STEP, Ch. 17
MODE, Ch. 17
CONTINUITY, Ch. 18
LORENTZ, Ch. 18

MAXWELL QUASI_NEUTRAL Command

Function: Specifies quasi-neutral electromagnetic algorithm.

Syntax: MAXWELL QUASI_NEUTRAL beta ;

Arguments: beta - plasma frequency and speed-of-light fraction, v/c (0<beta<1).

Defaults:

Defaults are set for all electromagnetic field algorithm parameters. The default algorithm is CENTERED. You can change this with one of the MAXWELL commands. You can use the TIME_STEP command to specify the time_step. If you do not specify the time_step, a default value equal to 80% of the centered-difference Courant criterion will be used, irrespective of the MAXWELL algorithm selected.

Description:

The MAXWELL QUASI_NEUTRAL command specifies a quasi-neutral solution of the fully time-dependent Maxwell's equations. The quasi-neutral algorithm is the same as centered-difference, but artificially reduces particle and displacement currents in Ampere's Law by a factor of beta squared. The result is a reduction in both the plasma frequency and speed-of-light by a factor of beta, and an increase in the Debye length by 1/beta. This allows the time step to be made larger by a factor of 1/beta, and this increase makes the algorithm suitable for use with very high density plasma situations, which would otherwise be impossible to model because of Courant, plasma-frequency, and Debye-length restrictions on the time step and grid size.

The quasi-neutral algorithm is most useful in a restricted class of problems in which magneto-hydro-dynamic (MHD) behavior is being investigated. Note that the Alfven velocity, $B^2/\mu_0\rho_m$, and collision-less skin-depth, c/ω_p, remain unchanged in this approximation, which indicates that ideal MHD behavior is preserved. In general, the quasi-neutral approximation may be used whenever plasma frequency oscillations and displacement current are neglected. The benefit of using QUASI-NEUTRAL is in the larger time step allowed, and the TIME_STEP command (Ch. 17) must be used to reset the time step to its larger value. The user must guarantee that the new time step does not exceed any of the following restrictions:

a) $\delta t < \left[\sum_{i=1,3} \frac{\beta^2 c^2}{\delta x_i^2}\right]^{-1/2}$, e.g., Courant condition with $c \rightarrow \beta c$,

b) $\delta t < 2[\beta\omega_p]$-1 , e.g., plasma frequency restriction with $\omega_p \rightarrow \beta\omega_p$, and

c) $\delta t < \delta x_{min} / v_{max}$, e.g., particle motion less than a single cell per time step,

where δx_{min} represents the minimum cell size and v_{max} represents the maximum particle velocity. The third restriction implies that the benefit of using QUASI_NEUTRAL is only possible if particles are non-relativistic with velocities less than about 20% of the speed of light, which corresponds to electrons below about 10 keV in energy. If electron energies exceed this level, the QUASI_NEUTRAL algorithm should not be used. Suggested values of beta are three to ten times the particle beta (v/c).

Restrictions:

1. This algorithm has an upper limit to the time step used.

2. It should be used only in simulations involving low velocity, nearly charge-neutral plasmas.

See Also: TIME_STEP, Ch. 17

Examples:

A simulation of MHD behavior in a 1 keV plasma might use quasi-neutral with a ten-fold increase in the default time step, thus decreasing the run time by a factor of 10.

```
AUTOGRID ;                           ! generates default SYS$DTIME variable
beta = 0.1                           ! beta is v/c
MAXWELL QUASI_NEUTRAL beta ;         ! switch to QUASI_NEUTRAL
dt = SYS$DTIME/beta ;                ! default is 80% of centered maximum
TIME_STEP  dt ;                      ! dt is now 800% of centered maximum
```

MAXWELL QUASI_STATIC Command

Function: Specifies quasi-static electromagnetic algorithm.

Syntax: MAXWELL QUASI_STATIC beta ;

Arguments: beta - speed-of-light fraction, v/c (0<beta<1).

Defaults:

Defaults are set for all electromagnetic field algorithm parameters. The default algorithm is CENTERED. You can change this with one of the MAXWELL commands. You can use the TIME_STEP command to specify the time_step. If you do not specify the time_step, a default value equal to 80% of the centered-difference Courant criterion will be used, irrespective of the MAXWELL algorithm selected.

Description:

The MAXWELL QUASI_STATIC command specifies a quasi-static solution of the fully time-dependent Maxwell's equations. The quasi-static algorithm is the same as centered-difference, but uses a speed-of-light artificially reduced by the factor, beta. This allows the time step to be made larger by a factor of 1/beta, and this increase makes the algorithm suitable for use with very low-velocity particles.

The quasi-static algorithm is useful in a restricted class of problems in which steady-state or very low frequency behavior is being investigated. In general, this would apply to simulations of either very low-velocity particles (v<<c) or very long-wavelength radiation (wavelength much larger than simulation dimensions). Such simulations are normally plagued by the need for excessively small time steps because of the Courant restriction in an electromagnetic simulation. The quasi-static algorithm introduces a factor in Faraday's Law that relaxes the Courant condition by slowing the speed-of-light propagation. Ampere's Law is not altered, so both the electrostatic and magnetostatic physical limits are preserved. However, the transients to the quasi-static limits are much slower in the simulation than in reality, and hence only quasi-static results are meaningful when this algorithm is employed.

The Courant stability criterion is given by $\beta\chi < 1$, where β is beta and

$$\chi^2 = c^2 \delta t^2 \sum_{i=1}^{N} \frac{1}{(\delta x_i)^2}.$$

is the Courant ratio squared, δx_i is the cell size in meters, and N is the number of dimensions (2 or 3). Certain geometries are even more restrictive (SYMMETRY AXIAL, Ch. 12). Since useful values of beta are less than unity, the allowable time_step is larger than the centered-difference time_step by the reciprocal of beta. The damping property of the quasi-static algorithm is the same as that for centered difference (the algorithm Q is infinite); however, the total absence of damping is not a problem for low-velocity particle applications.

To use the quasi-static algorithm, set beta to the desired fraction ($0< \beta <1$) of the speed-of-light. (Note that $\beta=1$ recovers the centered-difference algorithm.) The time_step may then be increased beyond the usual Courant limit by the reciprocal of this fraction. Suggested values of beta are three to ten times the particle beta (v/c), or 10 to 30 times the ratio of the simulation dimensions to the wavelength of the radiation.

Restrictions:

1. This algorithm has an upper limit to the time step used.

2. It should be used only in simulations involving no particles or very low-velocity particles.

See Also: **SYMMETRY AXIAL, Ch. 12**
TIME_STEP, Ch. 17
MODE, Ch. 17
CONTINUITY, Ch. 18
LORENTZ, Ch. 18

Examples:

An algorithm designed for very low-velocity particles in a 2D simulation might use quasi-static with a ten-fold increase in the default time step while suppressing the TE mode.

```
beta = 0.1 ;                        ! beta is v/c
MAXWELL QUASI_STATIC beta ;         ! switch to QUASI_STATIC
dt = SYS$DTIME/beta ;               ! default is 80% of centered maximum
TIME_STEP  dt ;                     ! dt is now 800% of centered maximum
MODE  TM ;                          ! suppress the TE mode
```

MAXWELL HIGH_Q Command

Function: Specifies high-Q electromagnetic algorithm.

Syntax: MAXWELL HIGH_Q [gamma [courant_ratio]] ;

Arguments:
gamma - filter factor (0 < gamma < 1).
courant_ratio - centered-difference Courant ratio squared (unitless).

Defaults:

Defaults are set for all electromagnetic field algorithm parameters. The default algorithm is CENTERED. You can change this with one of the MAXWELL commands. You can use the TIME_STEP command to specify the time_step. If you do not specify the time_step, a default value equal to 80% of the centered-difference Courant criterion will be used, irrespective of the MAXWELL algorithm selected.

The HIGH_Q algorithm provides defaults (which can be overridden) for gamma and courant_ratio, as described below.

Description:

The MAXWELL HIGH_Q command specifies a high-Q solution of the fully time-dependent Maxwell's equations. This algorithm artificially damps electromagnetic fields. It damps all frequencies in the same manner as the time-biased algorithm, but damps less at physical low frequencies, hence the name, "high-Q." It is especially suitable for cases involving relativistic particles and cavities.

The Courant stability criterion is given by

$$\chi < \left[\frac{2-3\gamma^2+2\gamma^3}{2\left(1-3\gamma^2+2\gamma^3\right)}\right]^{1/2}, \quad 0 \le \gamma \le 1/2$$

and

$$\chi < (3\gamma)^{1/2}, \quad 1/2 \le \gamma \le 1$$

where

$$\chi^2 = c^2 \delta t^2 \sum_{i=1}^{N} \frac{1}{(\delta x_i)^2}.$$

is the Courant ratio squared, δx_i is the cell size in meters, and N is the number of dimensions (2 or 3). Certain geometries are slightly more restrictive (SYMMETRY AXIAL, Ch. 12). In no case can the time_step exceed $\sqrt{3}$ times the centered-difference stability criterion. Also, it may not be possible to take advantage of the larger time_step due to constraints on outer boundaries (PORT, Ch. 12) and particle dynamics (LORENTZ, Ch. 18).

The damping function is given by

$$F(\beta^2) = \left(1-\gamma\beta^2\right)^2 \left(1+2\gamma\beta^2\right),$$

where β is the frequency, normalized to the highest value that can be supported by the spatial grid. In comparison with the time-biased algorithm, high-Q produces much less damping at lower frequencies, as shown in the following figure. However, note that some non-physical damping occurs at all non-zero frequencies. In other words, the algorithm itself has a Q that is frequency-dependent and must be accounted for in simulations involving saturation or energy balance.

You may adjust the degree of damping with the filter factor, gamma. A value of zero is equivalent to centered-difference, while a value of one causes maximum damping. The default and recommended value is 0.85. The high-Q algorithm derives the courant_ratio from the grid, but this value may be overridden by the user in rare cases, for example, when the cell with smallest grid is outside all simulation boundaries.

References:

MugShots, Vol. 1, No. 2 (May 1992).

Restrictions:

1. This algorithm has an upper limit to the time step used.

2. It is most appropriately used in relativistic particle applications where damping at low and intermediate frequencies would be undesirable, such as high-Q cavities.

3. Be aware that some damping occurs (the algorithm Q is finite) at all frequencies. This damping is completely unrelated to physical loss mechanisms such as resistivity or outgoing waves. In effect, the reciprocals of all Qs (including the algorithm Q) add to produce the effective Q.

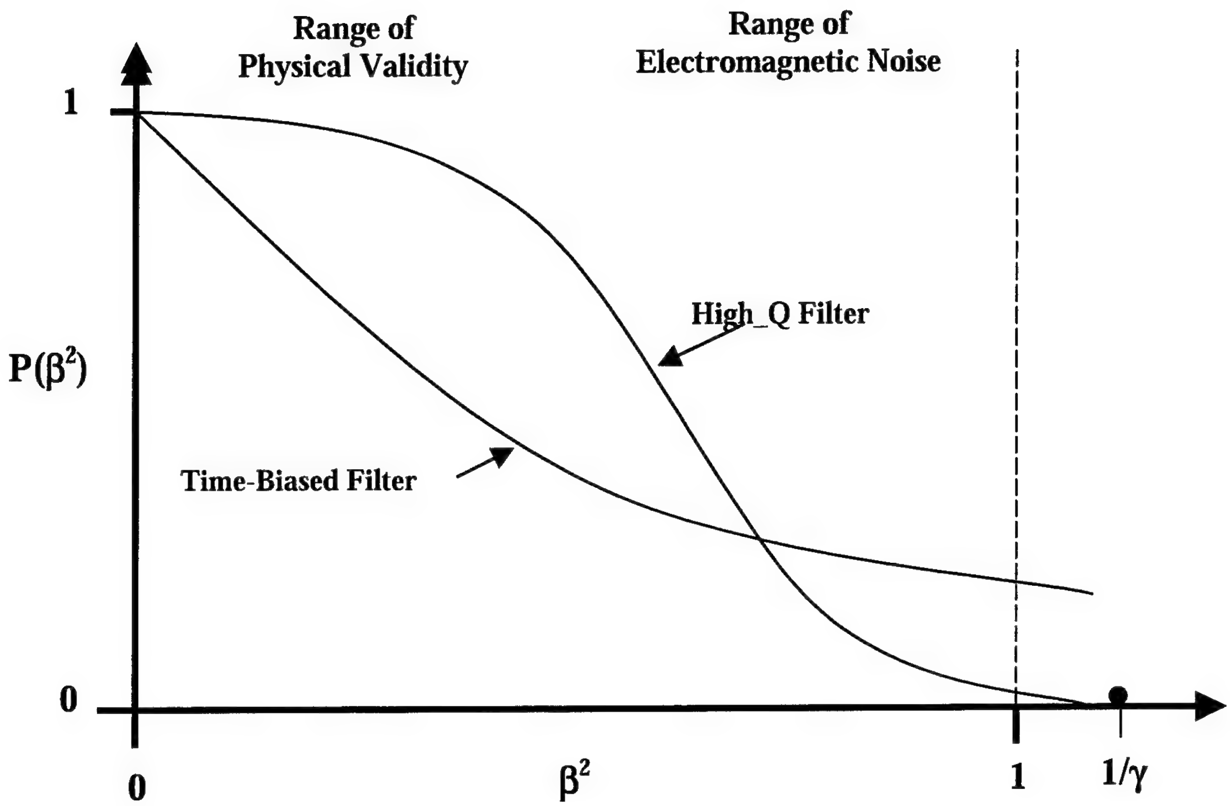
See Also: **SYMMETRY AXIAL, Ch. 12**
TIME_STEP, Ch. 17
MODE, Ch. 17
CONTINUITY, Ch. 18
LORENTZ, Ch. 18

Examples:

An algorithm designed for relativistic particles in a 2D simulation might use high-Q with a modest increase in the default time step while suppressing the TE mode.

```
MAXWELL  HIGH_Q ;               ! switch from BIASED to HIGH_Q
dt = SYS$DTIME * 1.5 ;          ! default is 80% of centered maximum
TIME_STEP  dt ;                 ! dt is 120% of centered maximum
MODE  TM ;                      ! suppress the TE mode
```

Note that a time step which exceeds the centered-difference maximum may lead to trouble with the particle algorithms (LORENTZ, Ch. 18) and outer boundaries (PORT, Ch. 12).



Comparison of time-biased and high-Q noise filters.

MAXWELL BIASED Command

Function: Specifies time-biased electromagnetic algorithm.

Syntax: MAXWELL BIASED [alpha1 alpha2 alpha3 [iterations [coefficient, ...]]] ;

Arguments:

alpha1	- forward-bias fraction (t+3/2dt).
alpha2	- center-bias fraction (t+1/2dt).
alpha3	- rearward-bias fraction (t-1/2dt).
iterations	- number of relaxation iterations (and coefficients).
coefficient	- relaxation coefficients (see Table).

Defaults:

Defaults are set for all electromagnetic field algorithm parameters. The default algorithm is CENTERED. You can change this with one of the MAXWELL commands. You can use the TIME_STEP command to specify the time_step. If you do not specify the time_step, a default value equal to 80% of the centered-difference Courant criterion will be used, irrespective of the MAXWELL algorithm selected.

The BIASED algorithm provides defaults (which can be overridden) for all parameters, as described below.

Description:

The MAXWELL BIASED command specifies a time-biased, iterative solution of the fully time-dependent Maxwell's equations. The time-biased algorithm is an implicit scheme designed to damp high-frequency noise arising from relativistic particles, poor particle statistics, or certain numerical instabilities.

The basic idea involves iterating between electric and magnetic field calculations during a single time step. Monotonically decreasing relaxation coefficients are applied to corrections in the electric field solution. Because of the iterations, this algorithm is computationally more expensive than the centered-difference algorithm, even though a larger time step can be used. The bias fractions represent contributions from magnetic field differences at three points in time: t+3/2dt, t+1/2dt, and t-1/2dt. These fractions are subject to the constraints,

$$\alpha_1 + \alpha_2 + \alpha_3 = 1$$
$$\alpha_1 > \alpha_3$$
$$\alpha_2^2 - 4\,\alpha_1\,\alpha_3 \geq 0.$$

The relaxation coefficients are typically in the range zero to one, with the first coefficient always equal to one. The following table lists various sets of temporal and relaxation coefficients, assuming $\alpha_3 = 0$.

The Courant stability criterion is given by $(\alpha_2^2 - 4\alpha_1\alpha_3)^{1/2}\chi < 1$, where

$$\chi^2 = c^2 \delta t^2 \sum_{i=1}^{N} \frac{1}{(\delta x_i)^2}.$$

is the Courant ratio squared, δx_i is the cell size in meters, and N is the number of dimensions (2 or 3). Some geometries are slightly more restrictive (SYMMETRY AXIAL, Ch. 12). Since the bias fraction factor is bounded, the allowable time_step is always greater than the centered-difference algorithm, and it can be substantially greater. Common choices using $\alpha_1 = \alpha_2 = 1/2$ (default), give double the centered-difference result. However, because the time-biased algorithm must iterate, it is always slower than centered difference. Also, it may not be possible to take advantage of the larger time_step due to constraints on outer boundaries (PORT, Ch. 12) and particle dynamics (LORENTZ, Ch. 18).

In the limit of sufficient iterations, the time-biased damping function is given by

$$F(\beta^2) = \left(1 + 4\alpha_1 \chi^2 \beta^2\right)^{-1},$$

where β is the frequency, normalized to the highest value that can be supported by the spatial grid. Thus, for reasonable choices of forward-bias fraction and Courant ratio, a high degree of damping of unwanted high frequencies associated with relativistic particles is achieved. However, note that some non-physical damping occurs at all non-zero frequencies. In other words, the algorithm itself has a Q, which is frequency-dependent and must be accounted for in simulations involving saturation or energy balance.

As indicated by the syntax, you may elect to let the code determine all of the optional values. In this case, the default values are α_1 = 1/2, α_2 = 1/2, α_3 = 0, iterations = 4, and coefficients = 1.0, 0.29912, 0.1502, 0.11111. If you want to supply only values for α_1, α_2, and α_3, the code automatically selects the minimum allowed value for iterations and generates the coefficients. You may elect to specify iterations as well; in this case, the code will generate the coefficients. Finally, you may enter all the optional values. See following table for typical values.

Coefficients for the time-biased algorithm.

α_1	I	τ's
0.125	2	1.0, 0.60494
	3	1.0, 0.75385, 0.60494
	4	1.0, 0.83944, 0.68410, 0.60494
0.25	2	1.0, 0.36000
	3	1.0, 0.52941, 0.36000
	4	1.0, 0.65759, 0.44305, 0.36000
0.50	4	1.0, 0.29912, 0.15022, 0.11111
	5	1.0, 0.39559, 0.20000, 0.13383, 0.11111
	6	1.0, 0.48267, 0.25457, 0.16470, 0.12613, 0.11111
0.75	8	1.0, 0.21488, 0.08768, 0.04944, 0.03359, 0.02591, 0.02205, 0.02041
	9	1.0, 0.25676, 0.10712, 0.60001, 0.04000, 0.03000, 0.02459, 0.02169, 0.02041
	10	1.0, 0.29857, 0.12791, 0.07159, 0.04717, 0.03472, 0.02775, 0.02371, 0.02144, 0.02041
	11	1.0, 0.33964, 0.14980, 0.08410, 0.05504, 0.04000, 0.03142, 0.02624, 0.02308, 0.02125, 0.02041
	12	1.0, 0.37943, 0.17256, 0.09743, 0.06355, 0.04579, 0.03551, 0.02919, 0.02517, 0.02262, 0.02111, 0.02041
	16	1.0, 0.52021, 0.26799, 0.15728, 0.10308, 0.07336, 0.05556, 0.04418, 0.03654, 0.03125, 0.02750, 0.02481, 0.02291, 0.02161, 0.02080, 0.02041

References:

B. B. Godfrey and B. Goplen, "Practical Evaluation of Time-Biased Electromagnetic Field Algorithms for Plasma Simulations," Mission Research Corporation Report, AMRC-N-146, presented at the Twenty-Second Annual Meeting APS Division of Plasma Physics, 10-14 November 1980.

Restrictions:

1. This algorithm has an upper limit to the time step used.

2. It is most appropriately used in relativistic particle applications to suppress high levels of numerical noise.

3. Be aware that some damping occurs (the algorithm Q is finite) at all frequencies. Thus, this algorithm is unsuitable for simulations involving resonant cavities with high values of Q, e.g., klystrons, etc. This damping is completely unrelated to physical loss mechanisms such as resistivity or outgoing waves. In effect, the reciprocals of all Qs (including the algorithm Q) add to produce the effective Q.

See Also: **SYMMETRY AXIAL, Ch. 12**
TIME_STEP, Ch. 17
MODE, Ch. 17
CONTINUITY, Ch. 18
LORENTZ, Ch. 18

Examples:

A 2D simulation designed for relativistic particles might use a time-biased solution with an aggressive increase in the default time step while suppressing the TE mode.

```
MAXWELL  BIASED ;               ! switch from BIASED to BIASED!
dt = SYS$DTIME * 2.0 ;          ! default is 80% of centered maximum
TIME_STEP  dt ;                 ! dt is now 160% of centered maximum
MODE  TM ;                      ! suppress the TE mode
```

Note that a time step which exceeds the centered-difference maximum may lead to trouble with the particle algorithms (LORENTZ, Ch. 18) and outer boundaries (PORT, Ch. 12).

This page is intentionally left blank.

18. CHARGED PARTICLES

This Chapter covers the following commands:

SPECIES		
LORENTZ		
CONTINUITY	**CONSERVED**	
CONTINUITY	**NOT_CONSERVED**	(2D simulations only)
CONTINUITY	**CORRECTED**	(2D simulations only)
CONTINUITY	**LOW_T**	(2D simulations only)

These commands specify the charged-particle algorithms. In 3D simulations, only the SPECIES and LORENTZ commands can be used. The CONTINUITY commands will not be accepted in 3D, since these algorithms are set entirely by default and cannot be altered. By contrast, 2D simulation offers more control over these algorithms.

You can use the SPECIES command to create new particle species, if required (electrons and protons are already available). You can use the LORENTZ command to specify how often the particle coordinates (X1, X2, X3) and momenta (P1, P2, P3) are calculated. The kinematics calculation is fully relativistic. You can use the CONTINUITY commands in 2D simulations to select the charge-continuity algorithm that calculates the charge-density field (QO) and the current-density fields (J1, J2, J3) from the particle motion. The charge-density field error (QERR) may also be calculated.

You may not need to use any of these commands. In both 2D and 3D simulations, the default charged-particle algorithms use the following parameters:

species available	- electrons and protons
Lorentz algorithm	- 3-D relativistic using the electromagnetic time_step
continuity algorithm	- CONSERVED (satisfies Gauss's law exactly)

This combination provides a very conservative, robust solution for simulations involving both nonrelativistic and highly relativistic particles. It is consistent with the electromagnetic algorithm defaults (Ch. 17). If you are unsure of your simulation requirements, this is the recommended configuration.

On the other hand, in a 2D simulation, you may elect to use the other particle algorithms to speed up the simulation, to improve fidelity by a better match of the algorithm to the physics, or even to suppress certain physical effects to gain insight. The following table provides guidance in selecting the continuity algorithm.

General conditions for continuity algorithm selection.

CONTINUITY ALGORITHM	CONSERVED	NOT_ CONSERVED	CORRECTED	LOW_T
Speed required		X		
Low noise required		X		X
Moderate noise acceptable			X	
Extreme noise acceptable	X			
Exact conservation required	X			
Approximate conservation acceptable			X	X
Poor conservation acceptable		X		
Beams and some plasmas	X		X	
Low-temperature plasmas		X		X

A more detailed explanation of the principal factors that influence the simulation follows.

1. Particle definition — Our "particle" is an artificial entity that typically represents a large number of physical particles. The charge-to-mass ratio for all particles of a given species is fixed and is usually, but not necessarily, identical with a physical species (see SPECIES command). An artificial particle can represent any number of physical particles to meet requirements of the emission processes, which include statistical properties such as creation frequency, particle number, etc. (EMISSION, Ch. 16). It is possible that no two particles will be identical in charge or mass, although the charge-to-mass ratio will be identical. Once created by an emission process, particles move through space under the influence of Lorentz forces (LORENTZ, Ch. 18) and contribute to charge and current densities (CONTINUITY, Ch. 18) which are used in Maxwell's equations (MAXWELL, Ch 17).

2. Time step — As with the electromagnetic field algorithms, there are temporal resolution requirements for particles. There is a time-step instability in the Lorentz and continuity algorithms which is analogous to the Courant instability in Maxwell's equations. To prevent it, particles must not be allowed to transverse more than one cell in a particle time step. By default, the particle time step is equal to the electromagnetic time_step, the default for which is 80% of the centered-difference Courant criterion. If both defaults are used, the particle stability requirement is automatically satisfied. However, if you use a more aggressive electromagnetic time_step and/or a non-unity kinematics multiplier, the stability requirement could be violated for relativistic particles. Other stability constraints which may be encountered include cyclotron frequency resolution, $\omega_c \delta t < 1$, an orbit resolution problem associated with high applied magnetic fields, and plasma frequency resolution, $\omega_p \delta t < 2$, a problem associated with high-density plasmas which can result in catastrophic instability.

3. Spatial grid — As with the electromagnetic field solution, accuracy in the particle orbits and current-density fields is determined primarily by the spatial resolution, i.e., the cell size (GRID and AUTOGRID, Ch. 11). For

example, if field resolution requirements dictate some maximum cell-to-cell longitudinal field variation (say, 25%), then a space-charge-limited boundary layer might be represented well by 30 cells, barely by 10 cells, and not at all by 3. There are also stability constraints which may be encountered such as Debye length resolution, $\delta x \leq \lambda_D$. This mild instability is encountered in low-temperature plasmas and results in "self-heating."

4. Local charge conservation — All of the continuity algorithms solve the equation

$$\nabla \cdot J + \partial_t \rho \quad = \quad 0$$

to obtain current- and charge-density fields from the particle motion. However, the algorithms differ in the way that they solve this equation and in the degree to which they satisfy Gauss's law,

$$\nabla \cdot E - \rho / \varepsilon \quad = \quad 0$$

locally, i.e., in each spatial cell. Unfortunately, the algorithm which provides exact charge conservation (conserved continuity) also produces the greatest particle noise. Nevertheless, because of the importance of satisfying Gauss's law, this is the recommended and default algorithm.

5. Noise — We typically represent a huge number of small, physical particles with a small number of huge, artificial particles. The motion of these huge particles in the finite spatial grid creates numerical noise in the electromagnetic fields. The magnitude of this noise goes inversely as the square root of the number of particles, so simulation improvement by this means alone can be an expensive proposition. Instead, some algorithms are designed specifically to reduce noise. Certain electromagnetic algorithms (HIGH_Q and BIASED) work directly on the electromagnetic fields by damping, while certain continuity algorithms (CORRECTED and LOW_T) reduce noise by working with the current-density fields.

6. Statistics — The issue is how to represent adequately the required phase space with the fewest particles. The emission processes (EMISSION, Ch. 16) provide statistical controls that allow particular regions of phase space to be emphasized with more particles (of lower charge). For highly correlated phase space (e.g., a charged-particle beam), relatively few particles are required. In such cases, particle methods have great advantage. On the other hand, a thermal plasma might require huge numbers of particles to represent even the simplest physics, and in such cases, fluid methods might be advantageous.

7. Dimensions — By default, the Lorentz kinematics are relativistic and three-dimensional. This is consistent with the electromagnetic field defaults (time-biased with both electromagnetic modes). However, 2D simulations which require only non-relativistic kinematics ($v/c < 0.2$) or two-dimensional kinematics can offer considerable speed advantage. Determining the relativistic requirement is usually straightforward. The dimensionality required can always be found by analyzing field (including any external fields) and particle momentum components and their interplay between Maxwell's equations and the Lorentz equation. Note that even if 3D kinematics are required, the component J3 may vanish from symmetry, so that a TE mode does not necessarily result.

8. Materials, etc. — All particle algorithms are compatible with the full range of material properties and outer boundaries. In general, particles may be created and destroyed by bulk property materials (Ch. 14), while they pass through unique structures (Ch. 15) which have infinitesimal thickness. At outer boundaries (Ch. 12), particles may have their coordinates or momenta altered (at symmetries) or they may be destroyed.

SPECIES Command

Function: Creates new particle species.

Syntax:

```
SPECIES  species
         CHARGE  charge_multiplier
         MASS   mass_multiplier { ELECTRON, PROTON, AMU } ;
```

Arguments:

species	- name of species, user-defined.
charge_multiplier	- electronic charge unit multiplier (signed and unitless).
mass_multiplier	- mass unit multiplier (unitless).

Defaults:

The ELECTRON and PROTON species have already been created and are ready for use. However, any other species that is required must be created explicitly.

Description:

The SPECIES command is used to create a new particle species. An internal species table contains the characteristics of all particle species to be used in the simulation. The ELECTRON and PROTON species reside permanently in the species table. You can add other species to the table, as required, using SPECIES commands. The particle species may be based upon physical particles (e.g., ELECTRON and PROTON), but there is no requirement that they be. You are free to create arbitrary particle species to satisfy simulation requirements, which can include non-physical particles such as heavier electrons or lighter protons. You can also create species with identical physical properties but different names to distinguish them in the output. Once created in the table, the new species can be used in a variety of emission (EMISSION, Ch. 16) and initialization (POPULATE, Ch. 19) processes.

A particle species is completely specified by its charge-to-mass ratio. However, for ease of identification with common physical particles, the SPECIES syntax has several arguments. You must give each new species a unique species name, which will be referred to in other commands. The electronic charge unit has a magnitude of e and positive sign. The CHARGE keyword is followed by the charge_multiplier, which is simply the number of units of electronic charge and carries the sign (+ or -). For convenience, several options are available to specify mass. The MASS keyword is followed by the mass_multiplier and the mass units, which may be ELECTRON, PROTON, or AMU. The AMU, or atomic mass unit (defined so that the mass of the carbon atom equals exactly 12 AMU) is more suitable for larger atoms and isotopes.

Restrictions:

The particle species table can contain up to fifteen (15) charged particle species.

See Also:

EMISSION processes , Ch. 16
POPULATE, Ch. 19
LORENTZ, Ch. 18

Examples:

1. A carbon isotope species can be quickly created with the command,

SPECIES carbon CHARGE +1 MASS 12 AMU ;

2. An electron species named "blue_electron" (to distinguish it from ELECTRON) can be created with the command,

SPECIES blue_electron CHARGE -1 MASS 1 ELECTRON ;

The "blue_electron" particles are identical with and behave in exactly the same manner as the "ELECTRON" particles. They simply have a different species name that allows them to be readily distinguished in phase-space plots, etc.

3. An artificial, light-weight species named "proton_lite" can be created with the command,

SPECIES proton_lite CHARGE +1 MASS 0.1 PROTON ;

At one-tenth the mass of physical protons, "proton_lite" particles will be much more responsive to electromagnetic forces. This might enable simulations precluded by a conventional approach, such as proton back-flow in an electron diode.

LORENTZ Command

Function: Specifies Lorentz kinematics algorithm.

Syntax: LORENTZ TIMING {step_multiple, periodic_timer} ;

Arguments:
step_multiple - multiple of electromagnetic time_step (integer).
periodic_timer - name of a periodic timer, defined in the TIMER command.

Description:

The LORENTZ command is used to specify the particle kinematics time step. The default is a step_multiple equal to 1.

This algorithm automatically adds magnetic fields from the Maxwell solution to any magnetostatic fields (PRESET, Ch. 19) and uses these to solve the Lorentz force equation to advance particle coordinates and momenta in time. The particle time step must be an integer step_multiple of the electromagnetic time_step (TIME_STEP, Ch. 17) to allow efficient simulation of low-velocity particles. However, the Lorentz algorithm has a stability criterion on the time step, and you must ensure that no particle is able to move more than one cell in a particle time step. If the default electromagnetic time_step and default step_multiple are used, stability is guaranteed; however, you may be able to improve efficiency with a more aggressive time_step and/or step_multiple.

The TIMER option may be used when you want to ensure a time delay before the onset of particle dynamics. In effect, you use a timer with the starting time set to the time at which you desire the particle algorithms to be activated. With a long delay time you may allow electromagnetic fields to be built up over a transient phase and then allow the activation of the standard particle algorithms.

The Lorentz algorithm calculates the particle coordinates (X1, X2, X3) and momenta (P1, P2, P3), which may be observed in various output commands (e.g., PHASESPACE, Ch. 24). The units of momentum are m/sec, which includes the relativistic factor, γ, but not the rest mass of the particle.

Restrictions:

1. The step_multiple (and electromagnetic time_step) must be chosen to preclude a particle from moving more than one cell width in a single particle time step.

2. The step_multiple must be the same in all LORENTZ commands.

3. Any timers that involve particle output must take the step_multiple into account. For example, PHASESPACE plots will be blank unless they are produced at the actual time of the particle kinematics.

See Also:
MAXWELL algorithms, Ch. 17
TIME_STEP, Ch. 17
MODE, Ch. 17
PRESET, Ch. 19
PHASESPACE, Ch. 24

CONTINUITY CONSERVED Command

Function: Specifies conserved continuity algorithm.

Syntax: CONTINUITY CONSERVED ;

Arguments: None.

Defaults:

Defaults are set for all particle algorithm parameters. The default continuity algorithm for both 2D and 3D simulations is CONSERVED. No other alternative is offered in 3D simulations, and no CONTINUITY command will be accepted. However, in 2D simulations, you can change the algorithm with one of the CONTINUITY commands.

Description:

The CONTINUITY CONSERVED command specifies a continuity algorithm which conserves charge (Gauss's law) exactly, but is high in numerical noise. It is suitable for most particle applications. An electromagnetic algorithm which provides damping, such as high_Q or time-biased, can be used to reduce noise if required (MAXWELL, Ch. 17).

This algorithm produces current-density fields which conserve Gauss's law exactly (within roundoff error). The algorithm can be derived from application of Gauss's law to conformal, rectilinear motion, with the particle orbit being a sum of such rectilinear motions and the order of the rectilinear motions being determined randomly. It requires no correction, since it achieves charge conservation directly. However, in PIC parlance, the algorithm uses "nearest-grid-point" charge allocation, and this is what produces the extreme numerical noise.

This algorithm calculates the charge-density field (QO) and the current-density fields (J1, J2, J3). Since no error in Gauss's law is produced, the charge-density error field (QERR) is not calculated for this algorithm.

Restrictions:

1. The step_multiple (and electromagnetic time_step) must be chosen to preclude a particle from moving more than one cell width in a single particle time step.

2. Only one CONTINUITY command can be given. (All particle species must use the same algorithm.) If more than one command is given, the last command will dominate.

3. If the CONSERVED algorithm is used with relativistic particles, then an electromagnetic field algorithm which includes damping, such as high-Q or time-biased, is recommended to reduce noise.

See Also: MAXWELL algorithms, Ch. 17
TIME_STEP, Ch. 17
MODE, Ch. 17
LORENTZ, Ch. 18

CONTINUITY NOT_CONSERVED Command

Function: Specifies not-conserved continuity algorithm in 2D simulations.

Syntax: CONTINUITY NOT_CONSERVED ;

Arguments: None.

Defaults:

Defaults are set for all particle algorithm parameters. The default continuity algorithm for both 2D and 3D simulations is CONSERVED. No other alternative is offered in 3D simulations, and no CONTINUITY command will be accepted. However, in 2D simulations you can change the algorithm with one of the CONTINUITY commands.

Description:

The CONTINUITY NOT_CONSERVED command specifies a continuity algorithm that is both fast and low in numerical noise. However, it does not conserve charge (Gauss's law) locally. It is suitable for brief, non-repetitive, transient phenomena, in which the errors (produced by the particles) in the electric fields are small in comparison with their magnitude.

This algorithm calculates the charge-density field (QO) and the current-density fields (J1, J2, J3). The error in Gauss's law,

$$\rho_{err} = \rho - \varepsilon \nabla \cdot E$$

is a scalar field named QERR which is accessible as output in the same manner as the other fields.

Restrictions:

1. The CONTINUITY NOT_CONSERVED command can be used only in 2D simulations.

2. The step_multiple (and electromagnetic time_step) must be chosen to preclude a particle from moving more than one cell width in a single particle time step.

3. Only one CONTINUITY command can be given. (All particle species must use the same algorithm.) If more than one command is given, the last command will dominate.

4. The NOT_CONSERVED algorithm should be used only where violations of Gauss's law will not adversely affect the results.

See Also:

MAXWELL algorithms, Ch. 17
TIME_STEP, Ch. 17
MODE, Ch. 17
LORENTZ, Ch. 18

CONTINUITY CORRECTED Command

Function: Specifies corrected continuity algorithm in a 2D simulation.

Syntax: CONTINUITY CORRECTED diffusion_coefficient ;

Arguments diffusion_coefficient - error diffusion coefficient (unitless).

Defaults:

Defaults are set for all particle algorithm parameters. The default continuity algorithm for both 2D and 3D simulations is CONSERVED. No other alternative is offered in 3D simulations, and no CONTINUITY command will be accepted. However, in 2D simulations, you can change the algorithm with one of the CONTINUITY commands.

Description:

The CONTINUITY CORRECTED command specifies a continuity algorithm which is low in numerical noise and conserves charge (Gauss's law) approximately. It uses the same algorithm as NOT_CONSERVED, but then it applies a correction which allows it to conserve charge (Gauss's law) approximately. The correction restores charge conservation at long time scales, as controlled by the diffusion_coefficient. A larger coefficient restores conservation more quickly, but results in greater noise.

In this algorithm, the error in Gauss's law,

$$\rho_{err} = \rho - \varepsilon \nabla \cdot E$$

is computed immediately following the electromagnetic field calculation. Then the subsequent current density is corrected by the addition of an error diffusion term,

$$\bar{J}' = \bar{J} + d\,\bar{\nabla}\rho_{err} \ ,$$

where $d = 1/4\ c_d\ (\delta x)^2/\delta t$ and c_d is the diffusion_coefficient, δx^2 is the minimum square grid spacing, and δt is the time step. (Use of this algorithm with a diffusion_coefficient of zero will produce the same result as the not_conserved algorithm, but at greater expense.) No benefit will accrue from using a diffusion_coefficient greater than 2. Useful values will lie between 0.1 and 1.

This algorithm calculates the charge-density field (QO) and the current-density fields (J1, J2, J3). The error in Gauss's law is a scalar field named QERR which is accessible as output in the same manner as the other fields.

References:

B. Goplen, R. Worl, and L. Ludeking, "A New Current Density Algorithm in MAGIC," Mission Research Corporation Report, MRC/WDC-R-155, December 1987.

Restrictions:

1. The CONTINUITY CORRECTED command can be used only in 2D simulations.

2. The step_multiple (and electromagnetic time_step) must be chosen to preclude a particle from moving more than one cell width in a single particle time step.

3. Only one CONTINUITY command can be given. (All particle species must use the same algorithm.) If more than one command is given, the last command will dominate.

4. The CORRECTED algorithm should be used only where local charge conservation is required but electromagnetic field damping is inadvisable.

See Also:

MAXWELL algorithms, Ch. 17
TIME_STEP, Ch. 17
MODE, Ch. 17
CONTINUITY NOT_CONSERVED, Ch. 18
LORENTZ, Ch. 18

CONTINUITY LOW_T Command

Function: Specifies low-T continuity algorithm in a 2D simulation.

Syntax: CONTINUITY LOW_T [debye_ratio] ;

Arguments: debye_ratio - Debye length-to-grid spacing ratio (unitless).

Defaults:

The default continuity algorithm for both 2D and 3D simulations is CONSERVED. No other alternative is offered in 3D simulations; other CONTINUITY commands are ignored. However, in 2D simulations, you can change the algorithm with one of the CONTINUITY commands.

Description:

The CONTINUITY LOW_T command specifies a continuity algorithm which conserves charge (Gauss's law) approximately and also reduces numerical noise. It is suitable for low-temperature plasma applications, where self-heating occurs due to failure of the spatial grid to resolve the Debye length adequately.

This algorithm makes use of the CONSERVED algorithm, which conserves charge (Gauss's law) exactly. To reduce the numerical noise associated with this algorithm, it then performs a temporal filter of the current-density fields. In the process, it loses the exact charge conservation feature. The algorithm has shown some ability to suppress the grid instability associated with particle self-heating when the Debye length (=thermal speed/plasma frequency) of a group of particles is smaller than the grid spacing. This generally occurs when debye_ratio is near the ratio of Debye length to the grid spacing.

$\lambda_d = v_{th} / \omega_p$.

This algorithm calculates the charge-density field (QO) and the current-density fields (J1, J2, J3). The charge-density error field (QERR) is not calculated. It is recommended that you use the MAXWELL BIASED option in conjunction with the LOW_T continuity selection in order to provide even greater cooling capability.

Restrictions:

1. The CONTINUITY LOW_T command can be used only in 2D simulations.

2. The step_multiple (and electromagnetic time_step) must be chosen to preclude a particle from moving more than one cell width in a single particle time step.

3. Only the last CONTINUITY command entered will be used. (All particle species use the same algorithm.)

4. The LOW_T algorithm should be used only in low-temperature plasma applications.

See Also: MAXWELL algorithms, Ch. 17
TIME_STEP, Ch. 17
MODE, Ch. 17
CONTINUITY CONSERVED, Ch. 18
LORENTZ, Ch 18

This page is intentionally left blank.

19. OTHER ALGORITHMS

This Chapter covers the following commands:

POPULATE (2D simulations only)
PRESET
COILS
DRIVER
CIRCUIT (2D simulations only)
POISSON (2D simulations only)
EIGENMODE

The group of commands involves initial conditions, external sources, and electrostatic and eigenvalue fields. The POPULATE, CIRCUIT, and POISSON commands can be used only in 2D simulations, and the rest can be used in either 2D or 3D. You can use the POPULATE command to create an initial charged-particle distribution and the PRESET command to initialize electromagnetic and magnetostatic fields. You can use the COILS command to generate a solenoidal magnetostatic field. You can use the DRIVER command to apply external current-density sources and the CIRCUIT command to apply external voltage sources between conductors. You can use the POISSON command to compute electrostatic fields and the EIGENMODE command to compute frequency eigenvalues and cyclic field distributions.

The following table provides guidance on the physical conditions that bear on the algorithm selection.

General conditions for algorithm selection.

CONDITION	POPULATE	PRESET	COILS	DRIVER	CIRCUIT	POISSON	EIGENMODE
External B		X	X				
External J				X			
External V					X	X	
Initial E, B		X					
Initial $\rho = 0$	X						
Initial $\rho \neq 0$	X	X				X	
Electrostatic $\rho = 0$						X	
Electrostatic $\rho \neq 0$	X					X	
Eigenvalues							X
Degenerate Modes		X					X

A brief discussion of the main considerations follows.

1. Initial charged-particle distributions — The POPULATE command can be used in 2D simulations to create an initial charged-particle distribution within a specified spatial area. If the resulting charge distribution is non-neutral, then electromagnetic field initialization is required. Recall that the default initial field vanishes, which implies that there is no net charge anywhere in space. Thus, if you create a charged particle but do not initialize

the fields to account for it, the code implicitly creates a second particle of opposite charge to cancel the one created. In effect, this second charge has infinite mass and remains embedded in the spatial grid for the duration of the simulation.

2. Initial field distributions — The PRESET command can be used to initialize electromagnetic fields (E1, E2, E3, B1, B2, B3) as well as electrostatic fields (E1ST, E2ST, E3ST, PHST) and magnetostatic fields (B1ST, B2ST, B3ST). As the simulation transient progresses, the electromagnetic fields will normally change from their initial values. The electrostatic and magnetostatic fields are presumed to be due to some external source, such as a permanent magnet, and these fields never change. In interpreting output, note that the dynamic electric fields (E1, E2, E3) may be initialized to electrostatic results (E1ST, E2ST, E3ST), but that the dynamic magnetic fields (B1, B2, B3) are never initialized to magnetostatic fields (B1ST, B2ST, B3ST). Instead, the magnetostatic and dynamic magnetic fields are automatically added together whenever particle forces are required for the Lorentz equation. This provides flexibility in the specification of magnetostatic fields. However, care should be taken that initialized electric fields are self-consistent, since charge may be present in the initial state of the system.

3. Magnetostatic fields — the COILS command can be used to generate magnetostatic fields due to coils. These fields are automatically entered in the magnetostatic field arrays (B1ST, B2ST, B3ST) and do not require PRESET.

4. External current-density sources — The DRIVER command can be used to specify external current-density sources which may be arbitrary functions of time and space. These sources will be applied directly to the electric field calculation and are not included in output of the current-density fields (J1, J2, J3), which are due solely to charged particles. Therefore, the DRIVER sources may not be observed, except though their effect on electromagnetic fields.

5. External voltage sources — The CIRCUIT command can be used in 2D simulations to apply an external voltage source between conductors that are otherwise unconnected. This may be used to maintain a specified voltage between conductors, and it is also commonly used to represent the effect of a transverse electromagnetic (TEM) wave propagating in the direction of the ignorable coordinate (x3). (Simulation of a magnetron presents a practical example.) A TEM waveform is always given by solution of the Laplacian, which we can obtain using a POISSON command. Note that the number of solutions required typically equals the number of unconnected conductors, minus one.

6. Electrostatic field solutions — In 2D simulations, there are two circumstances in which the POISSON command can be used to compute the static potential (PHST) and electrostatic fields (E1ST, E2ST). One circumstance application involves computing the initial electric field distribution for a non-neutral initial charge distribution. The charges are created with a POPULATE command, the electrostatic fields are computed with the POISSON command, and the dynamic fields (E1 and E2) are initialized using a PRESET command. The second circumstance involves determining the TEM waveform associated with propagation in the ignorable coordinate. In this case, there are no particles, but the various conductors will have specified potentials. The POISSON command is used simply to obtain a Laplacian solution. The voltages associated with these TEM fields are applied using the CIRCUIT command.

7. Eigenvalues and eigenfunctions — The EIGENMODE command can be used to obtain frequency eigenvalues and field eigenfunctions from a steady-state solution of Maxwell's time-dependent equations. No particle distributions may be used. The PRESET command can be used to initialize eigenfunctions in searching for degenerate modes.

POPULATE Command

Function: Creates an initial charged-particle distribution in a 2D simulation.

Syntax:

```
POPULATE  species  area  n_x1_sites  n_x2_sites
        { FUNCTION CHARGE q(x1,x2) p1(x1,x2) p2(x1,x2) p3(x1,x2) ,
          FUNCTION DENSITY rho(x1,x2) p1(x1,x2) p2(x1,x2) p3(x1,x2) ,
          READ CHARGE file record_tag q_record p1_record p2_record p3_record ,
          READ DENSITY file record_tag rho_record p1_record p2_record p3_record }
```

Arguments:

species	- name of particle species, defined in SPECIES command.
area	- name of a conformal spatial area, defined in AREA command.
n_x1_sites	- number of uniformly-spaced sites in x1 (integer).
n_x2_sites	- number of uniformly-spaced sites in x2 (integer).
q	- particle charge in Coulombs, spatial function or constant.
rho	- charge density in Coul/m^3, spatial function or constant.
p1	- x1-momentum in m/sec, spatial function or constant.
p2	- x2-momentum in m/sec, spatial function or constant.
p3	- x3-momentum in m/sec, spatial function or constant.
file	- name of file containing particle data.
record_tag	- value (usually time) used to select record.
q_record	- name of record containing charge values.
rho_record	- name of record containing charge_density values.
p1_record	- name of record containing x1_momentum values.
p2_record	- name of record containing x2_momentum values.
p3_record	- name of record containing x3_momentum values.

Description:

The POPULATE command creates an initial particle distribution. A particle is completely specified by its species, its charge, its coordinates (X1, X2, X3) and its momenta (P1, P2, P3). Note that the definition of momentum includes the relativistic factor, γ, but not the particle rest mass. The units are meters (or radians) for the coordinates and m/sec for momenta.

You must first specify the species and the area in which particles will be created. There will be n_x1_sites_x1 for particle creation, uniformly distributed in x1 within the area. Similarly, there will be n_x2_sites uniformly distributed in the other coordinate. Thus, the total number of particles created within area_name will be n_x1_sites times n_x2_sites.

The particle charge and momentum can be specified using one of four options: FUNCTION CHARGE, FUNCTION DENSITY, READ CHARGE, and READ DENSITY.

The FUNCTION options offer a very effective means of creating particles. The FUNCTION CHARGE option allows you to specify spatial functions for particle charge and three momentum components. The FUNCTION DENSITY option is exactly the same, except that the first function represents charge density instead of particle charge. Wherever the charge (or charge-density) function vanishes, no particles will be created. Thus, complicated spatial distributions can be created by properly constructing the charge function, making use of intrinsic functions such as STEP, MIN, MAX, RANDOM, and GAUSSIAN.

The READ options read particle data from a file. In the READ CHARGE option, you specify the file name and the record_tag followed by the names of records containing particle charge and the three momentum components. (The record_tag usually specifies the record time; if it is set to -1, then the first record will be used.) The READ DENSITY option is exactly the same, except that the first particle record contains density instead of charge data.

If the initial charge distribution is non-neutral, then field initialization using the POISSON and PRESET commands is required. Recall that the default initial field vanishes, which by Gauss's law implies that there is no net charge anywhere in space. Thus, if you create charged particles but do not initialize the fields to account for them, the code implicitly creates a second group of particles of opposite charge to cancel the ones created. In effect, particles in this second group have infinite mass and remain embedded in the spatial grid for the duration of the simulation.

See Also: **SPECIES, Ch. 18**
POISSON, Ch. 19
PRESET, Ch. 19

Restrictions:

1. The POPULATE command can be used only in 2D simulations.

2. The maximum number of POPULATE commands is 50.

References:

B. Goplen, "Simulations of Self-Colliding Orbit Stability Against Negative Mass Observed in Migma IV Experiment," APS Division of Plasma Physics, Baltimore, MD, 3-7 November 1986.

Examples:

The following commands create a fifty-by-fifty (2500) array of "electrons," each with a charge of 10^{-8} Coulombs and uniformly distributed in an area named "diode." The particle momenta are outward directed, increasing linearly from the origin.

```
FUNCTION  p1(x1,x2) = 1.e8 * x1 ;
FUNCTION  p2(x1,x2) = 1.e8 * x2 ;
FUNCTION  p3(x1,x2) = 0 ;
POPULATE  ELECTRON diode 50 50 FUNCTION CHARGE 1.E-8  p1  p2  p3 ;
```

PRESET Command

Function: Initializes specified electromagnetic, electrostatic, or magnetostatic field.

Syntax

```
PRESET  field
        { FUNCTION  function(x1, x2, x3)
          POISSON  poisson ,
          READ  file  data_type  file_field  file_format ,
          PANDIRA  file }
        [ MODIFY  { ADD, REPLACE } ]
        [ SCALE  factor ]
        [ SHIFT  x1  x2  x3 ] ;
```

Arguments:

field	-	field component (E1, E2, etc.).
function	-	spatial distribution, defined in FUNCTION command.
poisson	-	name of Poisson solution, defined in POISSON command.
file	-	name of the file containing field data.
data_type	-	data type (CONTOUR, LINPRINT, PERSPECTIVE, or SURFACE).
file_field	-	field component as recorded in the file.
file_format	-	file format (ASCII or BINARY).
factor	-	field scaling factor, constant or function defined in FUNCTION command.
x1, x2 ,x3	-	coordinate shift for field data in file.

Defaults:

The default initialization value for all electromagnetic, electrostatic, and magnetostatic fields is zero.

Description:

The PRESET command initializes a specified electromagnetic, electrostatic, or magnetostatic field. A separate command is required for each field component.

The only fields which can be initialized are the electromagnetic fields (E1, E2, E3, B1, B2, B3), the electrostatic fields (E1ST, E2ST, E3ST, PHST), and the magnetostatic fields (B1ST, B2ST, B3ST). As the simulation transient progresses, the electromagnetic fields will normally change from their initial values. The electrostatic fields never change during the simulation. They are normally used to initialize (PRESET) the dynamic electric fields or to modify them during the transient (CIRCUIT). The magnetostatic fields are presumed due to some external source, such as a permanent magnet, and these fields never change during the simulation. However, unlike the electrostatic fields, the magnetostatic fields cannot be used to initialize the dynamic magnetic fields. Instead, the magnetostatic and dynamic magnetic fields are added together only when particle forces are required for the Lorentz equation. This distinction is important to remember in interpreting the dynamic field values.

There are no constraints on the distributions that can be used for the magnetostatic fields, although we strongly advise making them divergence-free. Similarly, the electromagnetic field distributions must be self-consistent, i.e., satisfy Maxwell's equations. In particular, Gauss's law must always be satisfied. This is simple in the absence of charge (the default fields vanish) but requires solution of Poisson's equation (POISSON) to initialize electric fields (PRESET) if a non-neutral charge distribution (POPULATE) is created.

There are four options, or methods, of initializing fields: FUNCTION, POISSON, READ, and PANDIRA. The FUNCTION method requires that you define a function of the spatial coordinates. This function will be

evaluated precisely at the field locations to set the initial values. This method can be used to set any field in 2D or 3D simulations.

When particles are present in a 2D simulation, the POISSON method can be used to initialize the electric fields E1 and E2. (Note that space charge affects only the TM mode.) You must obtain a Poisson solution (the solution and the poisson name is specified in the POISSON command), which calculates the scalar field PHST and the electrostatic fields E1ST and E2ST. The PRESET command simply transfers the field values, e.g., E1ST to E1. (Note that E1ST and E2ST will never change, while E1 and E2 will evolve in time from their initial values.)

The READ method reads field values from the specified file and interpolates these values to the spatial grid. The input file must use the DUMP database format. Then data_type specifies the type of output, with the choices being CONTOUR, LINPRINT, PERSPECTIVE, or SURFACE. The file_field is the field component recorded in the file, and the format is either ASCII or BINARY. Directions for preparing an input file for use with the READ option are documented separately (see References).

The PANDIRA method reads and interpolates static magnetic field data directly from the output of the LANL Poisson Group codes. The input file name is the output file of the SF7 post processor, e.g., OUTSF7 (see Example 4 for more details). The magnetic field information in OUTSF7 is converted to the DUMP database format, and stored in two files, PANDRA01.FLD and PANDRA02.FLD. The first file contains the magnetic field for B1ST, and the second file contains the magnetic field for B2ST. The data type used is SURFACE. These files can be used in subsequent simulations in place of the OUTSF7 file. See Example 4.

You can use the MODIFY, SCALE, and SHIFT options to alter the fields being entered with PRESET. The REPLACE option can be used to patch together several contiguous spatial segments. For example, if data is available for three spatial areas, these may be read in and patched together. Use of REPLACE will cause the new field values to replace the previous values. If it is desired to create a superposition of fields, ADD should be used in place of REPLACE. With the ADD option, the fields are added to the existing fields established by earlier PRESET commands. The SCALE option and scale factor can be used to alter the magnitude of the entered field. The SHIFT option adjusts the spatial data from the input file, which may have used a different coordinate origin than the simulation.

Restrictions:

1. Each field component to be initialized requires a separate PRESET command.

2. In presetting dynamic fields (E1, E2, ...), care must be taken that the preset fields are self-consistent, since charge may be present in the initial state of the system and boundary conditions will be applied.

3. Caution must be used when initializing magnetostatic fields. Certain fields may not make sense geometrically in non-Cartesian coordinate systems, and furthermore, the use of a particular component can break an assumed symmetry chosen for the dynamic field solution. For example, an x1- or x2-component of magnetic field could drive the transverse electric (TE) mode due to particle motion in the x3 direction, and substantial error may result if the TE mode is not represented in the simulation.

4. In interpreting output, recall that dynamic magnetic fields do not include any magnetostatic fields. The latter are included only in the particle force calculation. The dynamic electric fields, by contrast, may have been initialized to electrostatic values or may have electrostatic components being introduced continuously.

5. If an initial particle distribution is used with default initial fields (zero), the algorithms will assume that fixed (immobile) charges are embedded in the grid to neutralize the initial distribution exactly. (The fixed charges remain after the particles begin to move.) The electric fields must be initialized to account for space charge properly.

See Also: **MAXWELL algorithms, Ch. 17**
LORENTZ, Ch. 18
POISSON, Ch. 19

References:

L. Ludeking, "Importing Field Data to MAGIC," MRC/WDC-M-037M, January 1990.

James H. Billen and Lloyd M. Young, "POISSON SUPERFISH," LA-UR-96-1834, Revised January 1997.

Examples:

1. The following commands are used to initialize the B3 fields to values from the function, "bfield."

```
FUNCTION bfield(x,y,z) = 0.1  * (x*x + y*y) ;
PRESET B3 FUNCTION bfield ;
```

2. The following commands are used to initialize the B3ST fields to previously computed values. The data is read from a file named "BCOIL." The data_type is LINPRINT, and the file_field in the file is B3. The data was recorded in ASCII format.

```
PRESET   B3ST   READ   BCOIL   LINPRINT   B3   ASCII ;
```

3. The following command is used to initialize the dynamic electric fields (E1 and E2) to values obtained from the Poisson solution, "Quick_Start." The Poisson solution is obtained for a charge distribution entered with the POPULATE command and with zero voltage on the "anode" and "cathode."

```
POISSON Quick_Start 2 cathode 0.0 anode 0.0 ;
PRESET E1 POISSON Quick_Start ;
PRESET E2 POISSON Quick_Start ;
```

4. The following commands will initialize the B1ST and B2ST fields with magnetic field data generated by the LANL Poisson Group PANDIRA code. If the USE_PANDIRA_FILE variable flag is set to 1, then this is the initial simulation, and the magnetic data is read directly from the OUTSF7 file and translated into the MAGIC database format. The results are stored in the two data files PANDRA01.FLD and PANDRA02.FLD. In the event that you have already run this simulation previously, you may set the flag to 0 and read the translated database files directly. In both cases, the SHIFT keyword is used to introduce a translation of the data long the x1-axis. This may be necessary to align the fields properly with the simulation geometry absolute coordinates.

```
USE_PANDIRA_FILE = 1 ;
IF (USE_PANDIRA_FILE.EQ.1) THEN ;
      PRESET B1ST PANDIRA  OUTSF7 SHIFT -.541 0.;
      PRESET B2ST PANDIRA  OUTSF7 SHIFT -.541 0.;
ELSE ;
      PRESET B1ST READ PANDRA01.FLD SURFACE B1ST ASCII SHIFT -.541 0.;
      PRESET B2ST READ PANDRA02.FLD SURFACE B2ST ASCII SHIFT -.541 0. ;
ENDIF;
```

5. An OUTSF7 file is generated by the LANL Poisson Goup SF7 post-processor program. It is run after PANDIRA completes its solution. Typically, the OUTSF7 file contains lines similar to the following:

```
Program SF7.  Starting from TAPE35 dump 1.
Memory used for the TAPE35 arrays:      2.286 M

Magnetic fields for a rectangular area with corners at:
(Rmin, Zmin) = (   0.00000,   20.00000)
(Rmax, Zmax) = (   1.00000,   30.00000)
R and Z increments:     10    100

     R          Z           Br            Bz             |B|            A
   (in)       (in)          (G)           (G)            (G)          (G*cm)
 0.000000   20.00000  0.000000E+00 -1.212405E+02  1.212405E+02  0.000000E+00
 0.100000   20.00000  4.808673E+00 -1.210583E+02  1.211538E+02 -1.538582E+01
 0.200000   20.00000  9.601525E+00 -1.205166E+02  1.208984E+02 -3.070239E+01
 0.300000   20.00000  1.436256E+01 -1.196182E+02  1.204774E+02 -4.588178E+01
 0.400000   20.00000  1.907654E+01 -1.183645E+02  1.198920E+02 -6.085615E+01
 0.500000   20.00000  2.372917E+01 -1.167577E+02  1.191446E+02 -7.555807E+01
 0.600000   20.00000  2.830732E+01 -1.147998E+02  1.182383E+02 -8.992052E+01
 0.700000   20.00000  3.279901E+01 -1.124920E+02  1.171761E+02 -1.038767E+02
 0.800000   20.00000  3.719333E+01 -1.098341E+02  1.159607E+02 -1.173596E+02
 0.900000   20.00000  4.147977E+01 -1.068228E+02  1.145936E+02 -1.303017E+02
 1.000000   20.00000  4.564719E+01 -1.034509E+02  1.130741E+02 -1.426337E+02
 0.000000   20.10000  0.000000E+00 -1.312303E+02  1.312303E+02  0.000000E+00
 0.100000   20.10000  5.179233E+00 -1.310368E+02  1.311392E+02 -1.665381E+01
 0.200000   20.10000  1.034080E+01 -1.304619E+02  1.308711E+02 -3.323413E+01
 0.300000   20.10000  1.546807E+01 -1.295091E+02  1.304296E+02 -4.966889E+01
... etc ...
```

This sample data was produced by running AUTOMESH, LATTICE, PANDIRA, and then SF7 with the following input file, IN7.

```
RECT NOSCREEN
0.0 20 1.0 30.0
10 100
END
```

COILS Command

Function: Defines solenoidal magnetostatic field coils.

Syntax: COILS { AXIS { X1, X2, X3 },
DEFINITION coil_center, coil_radius, coil_current, ... } ;

Arguments:

coil_center - position of individual coil center, coordinates or name of point defined in POINT command.
coil_radius - radius of individual coil (m).
coil_current - current in individual coil (amps).

Description:

The COILS command specifies a set of coils used to generate a solenoidal magnetic field in a 3D simulation. Note: the axial magnetic field at the center of a coil is given by

$$B_{zo} = \mu_o I/2R,$$

where I is current and R is the coil radius and μ_o is the vacuum permeability.

You must enter one COILS command with the AXIS option to specify the axis of alignment for the coils. In CARTESIAN coordinates, this axis can be parallel to any of the coordinate axes (X1, X2, or X3), but may be offset from the origin. However, in POLAR coordinates, the coil axis must coincide with the simulation axis (X3). All coils are assumed to be aligned along the same axis; if more than one AXIS option is entered, all but the last will be ignored.

The DEFINITION option can be used to enter information on individual coils by giving the spatial position of the center the radius, and the current for each coil. Data for all of the coils may be entered in a single command.

Magnetostatic fields generated by COILS commands are automatically stored in the magnetostatic field arrays (B1ST, B2ST, B3ST) and do not require the use of the PRESET command. They are superimposed on any magnetostatic fields generated with the PRESET command. Note that magnetostatic fields are never used to PRESET the dynamic magnetic fields. Instead, the magnetostatic and dynamic magnetic fields are automatically added together when required for the particle force calculation. This is important to remember in interpreting magnetic field data.

Restrictions:

1. A maximum of 200 coils may be specified.

2. The physical location of coils should be outside of the dynamic simulation volume. In particular, do not place coils in locations where particle trajectories can intersect the coils themselves. This will result in unphysically large magnetic forces.

3. In interpreting output, recall that dynamic magnetic fields do not include any magnetostatic fields. The latter are included only in the particle force calculation. The dynamic electric fields, by contrast, may have been initialized to electrostatic values or may have electrostatic components being introduced continuously.

See Also: **PRESET, Ch. 19**

Examples:

This example illustrates a single coil centered on the origin, with axis aligned with x3.

```
COILS AXIS X3 ;
radius = 10mm ;
current = 2.*Bguide*Radius_COil/1.2566e-6 ;
COILS  DEFINITION 0,0,0  radius, current ;
```

DRIVER Command

Function: Applies an external current-density to a spatial object.

Syntax: DRIVER { J1, J2, J3 } j(t,x_1,x_2,x_3) object [CIRCUIT time_constant desired(t) obs$name] ;

Arguments:

j	- name of current-density function, defined in FUNCTION command.
object	- name of spatial object, defined in POINT, LINE, AREA, or VOLUME command.
time_constant	- circuit time constant (sec).
desired	- desired value of the observe measurement.
obs$name	- observe measurement.

Description:

The DRIVER command specifies a prescribed (analytic) current-density source to drive electromagnetic fields in a local region of space. The current-density component is either J1, J2, or J3. The current density function, in units of A/m^2, must be specified using the FUNCTION command.

The CIRCUIT option introduces a feedback loop that rescales the driving function, j. It compares the desired value with the measurement specified by the obs$name variable. The adjustment to the rescaling function is mediated by the time_constant according to:

$$\text{rescale}(t) = \exp\left[\frac{1}{\delta t}\left(1-\frac{\delta t}{\tau}\right)\int_0^t\left(1-\frac{\text{obs\$name}(t')}{\text{desired}(t')}\right)dt'\right] \quad ,$$

where δt is the time step. Most geometries have a practical lower limit on the time_constant, τ, which is typically a round-trip transit time, or perhaps a couple periods of an oscillation. The user must determine this practical limit using a combination of physical intuition, common sense, experience, and trial-and-error. Attempting to set the time_constant below the practical limit will result in dramatic overshoot of the desired behavior, oscillation, and possibily even instability. A typical use for the CIRCUIT option would be to specify a current-density source of a particular desired power, rather than current. In such a case, the obs$name measurement would come from an OBSERVE FIELD_POWER E.J_DRIVER.DV command. Frequently the current-density source is oscillatory so that the feedback adjusts the amplitude in such a manner as to arrive at some desired cycle-averaged quantity, such as power. <u>In such a case, it is essential to use the FILTER STEP option in the OBSERVE command to arrive at an appropriate cycle-averaged measurement</u>.

The current-density source is applied directly to Ampere's Law, and does not appear in diagnostics of the particle current-density fields J1, J2, and J3. Instead, the DRIVER current may be diagnosed with the OBSERVE FIELD_INTEGRAL J_DRIVER.DA command, and the DRIVER power may be diagnosed with the OBSERVE FIELD_POWER E.J_DRIVER.DV command.

Restrictions:

1. The spatial object must be conformal.

2. When the CIRCUIT option is used, the OBSERVE command referenced with the obs$name argument must employ the FILTER STEP tperiod option.

See Also: **MAXWELL algorithms, Ch. 17**

Examples:

1. To apply the current density source,

$$J_3\,(t,x_1) = 100x_1 \cos(2\pi 10^9 t)\ ,$$

within an area labeled "source_area" in a 2D simulation, one would use the commands,

```
FUNCTION source(t,x1) = 100 * x1 * COS(6.28E9*t) ;
DRIVER J3 source source_area ;
```

2. To apply a current density source that provides a specific amount of CW power to the simulation in the region called JUNCTION, you use the CIRCUIT option. The following commands are taken from a simulation in a coaxial geometry in cylindrical (z,r,phi) coordinates. Please note that the suffix used in the OBSERVE command is referenced in the CIRCUIT option of the DRIVER command. This is a mandatory structure. Note: the OBSERVE command also employs a FILTER STEP option. This is required since the CIRCUIT option requires the mean power flow, not the instantaneous power flow.

```
FUNCTION RAMP(T) = STEP(T,TPERIOD2)+STEP(TPERIOD2,T)*0.5*(1-COS(T*WFREQ2));
FUNCTION DESIRED_PWR(T) = 10watts*Ramp(t) ;
FUNCTION JDRIVER(T,Z,R) = 1/R*COS(T*WFREQ)*RAMP(T) ;
DRIVER J2 JDRIVER JUNCTION
        CIRCUIT TPERIOD2 DESIRED_PWR OBS$DRIVER_PWR ;
OBSERVE FIELD_POWER E.J_DRIVER.DV  JUNCTION  ! Basic observe command.
        FILTER STEP TPERIOD2                 ! Filter option employed.
        SUFFIX DRIVER_PWR ;                  ! Suffix name option employed.
```

CIRCUIT Command

Function: Specifies an external circuit to apply voltage between conductors in a 2D simulation.

Syntax:

```
CIRCUIT   v(t)   poisson
          [ MEASURE   { MATRIX  area, INTEGRAL  lines [ line,... ] } ]
          [ MODEL    { STATIC, RESISTIVE  z, INDUCTIVE  z  l } ] ;
```

Arguments:

v(t)	- voltage function, defined in FUNCTION command.
poisson	- Poisson solution name, defined in POISSON command.
area	- name of spatial area, defined in AREA command.
lines	- number of voltage measurements (integer).
line	- name of spatial line, defined in LINE command.
z	- circuit impedance (ohms).
l	- circuit inductance (henrys).

Defaults:

The defaults are MEASURE MATRIX with the full simulation area and MODEL STATIC.

Description:

The CIRCUIT command provides a means of applying a time-varying voltage between a set of unconnected conductors in a 2D simulation. Physically, this can be used to represent a transverse electromagnetic (TEM) wave traveling in the third (symmetry) spatial dimension. In a magnetron simulation, for example, the dominant field and particle kinematics occur in the (r,θ) plane, but the voltage pulse between anode and cathode propagates in the axial (z) coordinate. This can be approximated using the CIRCUIT command. The electrostatic solution for the TEM wave in the third dimension is computed using the POISSON command, which must be used if the CIRCUIT command is given.

The temporal voltage function, v(t), represents the external voltage source which is applied to the circuit connecting two or more conductors. The TEM solution is identified in both commands by the poisson name, which is defined in the POISSON command.

The keyword MEASURE specifies the method of measuring the voltage between pairs of conductors. The MATRIX option specifies the capacitive matrix method to obtain voltages. The dot product of the dynamic electric field and electrostatic field solutions over the area is taken, and voltages are computed dynamically on the basis of energy conservation by solving the coupled capacitance equations. The INTEGRAL option provides line-integral measurements between two conductors to obtain a voltage. The argument line specifies the number of voltage measurements, each consisting of a straight-line integral along each line. The INTEGRAL method is much faster than the MATRIX method, since it depends only on a few line integrals rather than complete volume integrals.

The keyword MODEL is used to specify one of three circuit models: STATIC, RESISTIVE, and INDUCTIVE. In the STATIC model (zero impedance), the conductor voltage is maintained precisely at the value specified by the external voltage source. In the RESISTIVE model, the source voltage is applied through the finite external impedance specified by z. In the INDUCTIVE model, the source voltage is applied through the finite external impedance and inductance specified by z and l, respectively. The default model is STATIC.

For applications in which the applied voltage across different pairs of conductors is varied independently, it is necessary to specify multiple Laplacian solutions and circuits. For example, in a triode consisting of a cathode, grid, and anode, the voltage on the grid may be varied with an RF signal, whereas the anode voltage might be held fixed. In this case, two Laplacian solutions and two circuits are required.

The circuit algorithm calculates variables that can be recorded by using OBSERVE commands. The following table lists the variable names and the physical symbol and definition associated with each.

Variable	Symbol	Definition
DCHARGE	δQ	differential charge added by circuit
CHARGE	Q	total charge added by circuit
CURRENT	I	current in circuit
DENERGY	δE	differential energy added by circuit
ENERGY	E	total energy added by circuit
POWER	P	circuit power added
VOLTAGE	V	applied circuit voltage
DVOLTAGE	δV	differential voltage added to system

Restrictions:

1. The CIRCUIT command is available only in 2D simulations.
2. The maximum number of CIRCUIT commands is five.
3. All circuits must use the same measurement method.

See Also: POISSON, Ch. 19
OBSERVE, Ch. 22

References:

L. Ludeking, B. Goplen, W. M. Bollen, "Gated Emission Simulations," Mission Research Corporation Report, MRC/WDC-R-119, August 1986.

L. Ludeking, B. Goplen, and N. Vanderplaats, "Simulation of Gated Emission Triodes," Bulletin of the American Physical Society, Vol. 31, p. 1600, November 1986.

Example:

A typical use of the POISSON and CIRCUIT commands is in introducing the voltage pulse in cross-field devices such as magnetrons. In this example, the circuit algorithm uses the function "VAPPLY" for the external voltage source and applies the Laplacian solution with the name "ESTATIC." The STATIC circuit model is used by default. The voltage measurements are also made by default using the MATRIX method, with the area being the complete simulation area. Figure (a) below shows the electron trajectories resulting when the static field is used to apply a voltage and a guiding axial magnetostatic field is also present. Figure (b) below shows the contours of the radial electric field.

```
POISSON ESTATIC 8 CATHODE 0.0
      ANODE.SHELL 1.0
      ANODE.VANE1 1.0
      ANODE.VANE2 1.0
      ANODE.VANE3 1.0
      ANODE.VANE4 1.0
      ANODE.VANE5 1.0
      ANODE.VANE6 1.0
      INITIAL X1
      TEST 5000 10 1.E-5
      ALGORITHM SCA 1 ;
FUNCTION VAPPLY(T) = AK.VOLT*(1.0-EXP(-T/TRISE)) ;
CIRCUIT VAPPLY ESTATIC ;
```

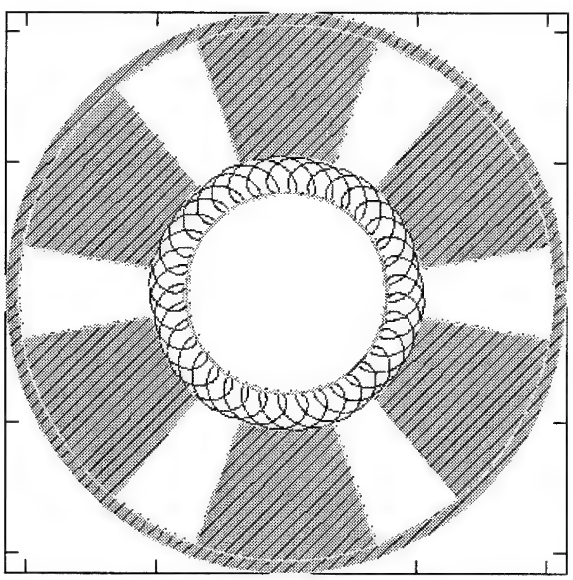


Figure (a).

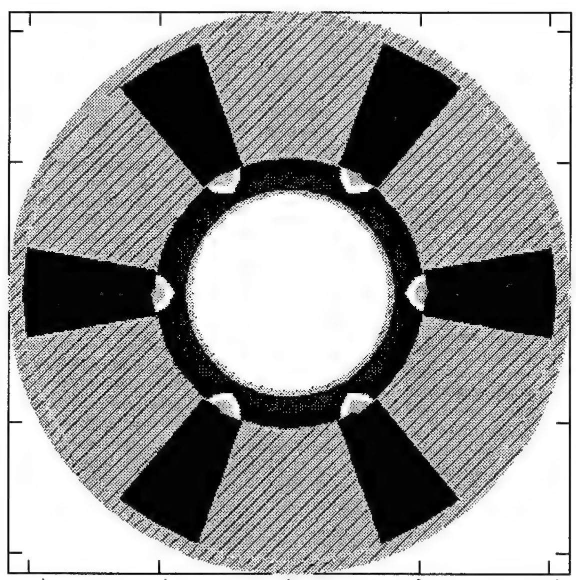


Figure (b).

Figure (a) shows electron trajectories, and Figure (b) shows the radial electric field contours.

POISSON Command

Function: Specifies an electrostatic solution in a 2D simulation.

Syntax:

```
POISSON poisson  conductors  area,voltage ...
        [ ALGORITHM { SOR omega,  SCA radius, ADI i1flag i2flag } ]
        [ TEST total_iterations  test_iterations  test_error  ]
        [  INITIALIZE  function(x1, x2) ]
        [ POPULATE  scale_factor ]  ;
```

Arguments:

poisson	- Poisson solution name.
conductors	- number of unique conductors.
area	- name of object, defined in AREA command.
voltage	- object potential (V).
omega	- over-relaxation coefficient (unitless).
radius	- spectral radius of the Jacobi matrix (unitless).
i1flag	- x1-coordinate ADI iteration flag (= 0, 1, or 2).
i2flag	- x2-coordinate ADI iteration flag (= 0, 1, or 2).
total_iterations	- maximum number of iterations (integer).
test_iterations	- iterations between convergence tests (integer).
test_error	- convergence criterion (unitless).
gradient	- initial potential function (NULL, READ, X1, X2 or defined in FUNCTION command.
scale_factor	- scale factor to apply to initial charge density.

Defaults:

The POISSON command must be given explicitly with specified voltages to obtain a Poisson solution (there is no default command). For the options, the defaults are gradient = NULL, scale_factor = 1, total_iterations = 10,000, test_iterations = 10, and ALGORITHM SOR with omega = 1.

Description:

The POISSON command is used to specify a solution of the generalized Poisson equation to obtain an electrostatic field distribution in a 2D simulation. It will produce values for the scalar potential (PHST) as well as the electrostatic field (E1ST, E2ST). This solution may be used to construct an initial condition with space charge (POPULATE, Ch. 19) for a subsequent transient simulation or to represent a TEM wave propagating in the third (symmetry) coordinate (CIRCUIT, Ch. 19).

You must specify the Poisson name, since it is possible to have more than one Poisson solution. This is followed by specifying the number of conductors followed by a list of object names and voltages.

The numerical solution is obtained by one of three methods: successive over-relaxation (SOR), SOR with Chebyshev acceleration (SCA), or alternating direction implicit (ADI). The SOR algorithm uses the parameter omega to improve the rate of convergence. When omega is between one and two, the algorithm provides over-relaxation. When omega is between zero and one, the algorithm is called under-relaxation. For some problems, under-relaxation may provide superior convergence properties.

The SCA algorithm is distinct from the SOR algorithm as it uses a variable over-relaxation coefficient omega. The spectral radius is used to calculate the values of omega on each iteration cycle. SCA reduces to SOR (omega=1) for a radius of 0. If the spectral radius is not known, it can be estimated from the expression

$$\rho = \frac{\cos(\pi / \mathrm{I1MX}) + (\mathrm{dx}/\mathrm{dy})^2 \cos(\pi / \mathrm{I2MX})}{1 + (\mathrm{dx}/\mathrm{dy})^2} .$$

This equation assumes a rectangular I1MX x I2MX grid, and allows for $dx \neq dy$. This expression holds for homogeneous Dirichlet and Neumann boundary conditions. For periodic boundaries, replace π by 2π. For large values of I1MX and I2MX, ρ is usually close to unity, and this can be used as a starting estimate. Chebyshev acceleration always provides improved convergence behavior over the standard SOR algorithm.

Generally, the ADI algorithm provides the most rapid convergence. The iteration flags, i1flag and i2flag, determine which coordinate is implicit. For entries of 0 and 0, both directions are implicit. Alternatively, entries 1 or 2 describe operations on a coordinate that is not implicit in a given iteration; either a single pass or a double pass (first odd, then even) will be performed.

All three methods require iteration, and a convergence test is performed at test_iteration intervals. Failing convergence to achieve test_error, the maximum number, total_iterations, will be performed. These parameters may be altered from their default values using the TEST keyword.

The keyword INITIALIZE is used to select the type of initialization function to be used. The options here are NULL, for no initialization, or X1 or X2, which specify the coordinate of maximum gradient. (Proper choice of arguments, X1 or X2, will enhance convergence.) Alternatively, a function name can be specified, and in this case, the potential is initialized to a two-dimensional field specified by the function. If READ is specified, then the initialization values are read in through the use of the PRESET PHST READ... command (PHST is the name of the scalar potential).

The keyword POPULATE is used to enter the scale_factor to be applied to the initial charge distribution (POPULATE, Ch. 19).

Restrictions:

1. The POISSON command can be used only in 2D simulations.

2. The treatment of axial symmetry for the spherical coordinate system is approximate and suitable only for large radii.

3. The origin is not treated in the polar (r,θ) coordinate system.

4. Conductors specified as unique potential surfaces must be so defined in the CONDUCTOR command. No more than ten conductors can be referenced in the POISSON command.

5. The over-relaxation coefficient omega is convergent only for $0 \leq$ omega ≤ 2. If $0 \leq$ omega ≤ 1, then under-relaxation is being used. For some problems, under-relaxation provides superior convergence.

6. The Jacobi spectral radius is restricted to values of $0 \leq$ radius ≤ 1.

7. The convergence criterion, test_error, should be set to order 10^{-5} for single precision calculations on a 32-bit machine.

See Also: **AREA, Ch. 10**
CIRCUIT, Ch. 19
POPULATE, Ch. 19
PRESET, Ch. 19

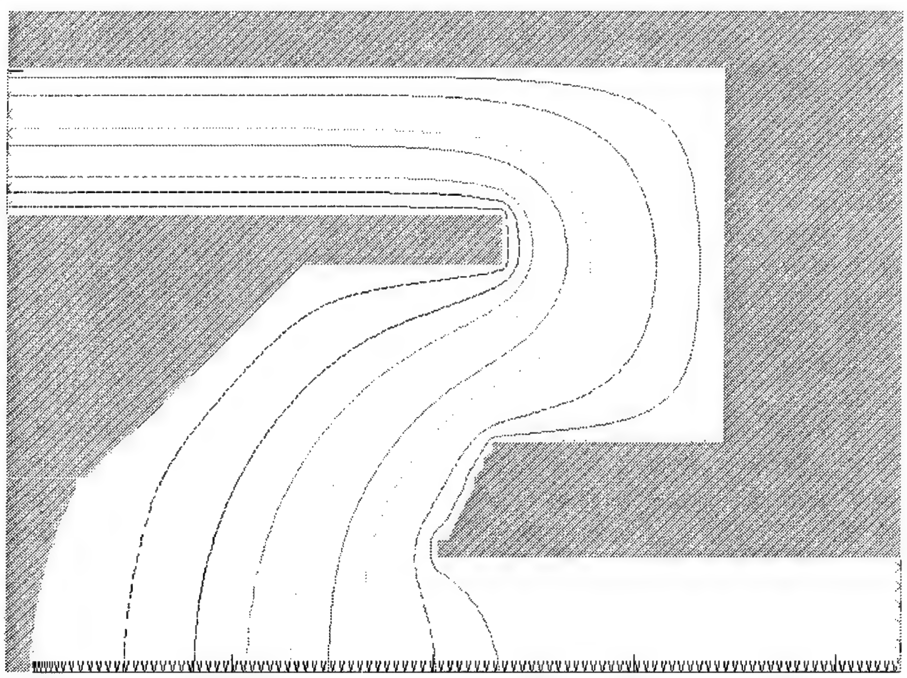
References:

B. Goplen, W. M. Bollen, J. McDonald, I. Coleman, and R. E. Clark, "Solution of the Generalized Poisson's Equation in MAGIC," Mission Research Corporation Report, MRC/WDC-R-049, February 1983.

Example:

The following command instructs the code to obtain a Poisson solution labeled PierceGun in a 2D simulation having three conductors, named cathode, focus and anode, in which the anode is at 1 volt with respect to the cathode and focus. The remaining arguments use default values. The following figure displays the equipotential contours.

```
POISSON PierceGun 3 CATHODE 0.0 FOCUS 0.0  ANODE 1.0;
```



This figure shows equipotential contours for a Pierce gun.

EIGENMODE Command

Function: Specifies eigenmode electromagnetic algorithm.

Syntax:

```
EIGENMODE
        [ SCAN_AT frequency ]
        [ SCAN_FROM frequency ]
        [ SCAN_TO frequency ]
        [ SCAN_LIST scan_number frequency, frequency, ... ]
        [ WINDOW df(f) ]
        [ ENERGY_REGION { area, volume } ]
        [ SAFETY_FACTOR safety_factor ]
        [ ITERATIONS iterations ]
        [ NOPRESET ]
        [ MODE { TE, TM, BOTH } ] ;
```

Arguments:

frequency	- center frequency of eigenmode test (Hz).
scan_number	- number of discrete frequency scans (integer).
df(f)	- width of frequency window (Hz), constant or defined in FUNCTION command.
area	- name of spatial area, defined in AREA command (2D simulation).
volume	- name of spatial volume, defined in VOLUME command (3D simulation).
safety_factor	- safety factor (unitless, default is 1.15).
iterations	- polynomial applications (integer, default is 30.)

Defaults:

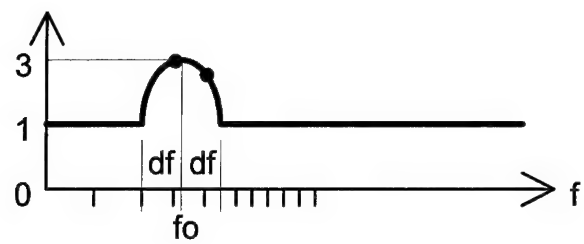
The algorithm parameters are completely defined using the following defaults. The SCAN_AT frequency is zero (seeks the lowest mode). The value of df(f) is derived automatically from the simulation dimensions. The area_name is the entire simulation area, the safety_factor is 1.15, and the iterations is 30. The MODE option applies only to 2D simulations; the default mode is BOTH.

Description:

The EIGENMODE command specifies an eigenvalue solution of the fully time-dependent Maxwell's equations. It can be applied to 2D or 3D simulations and to any geometry consisting of non-lossy elements and models, e.g., conductors, symmetries, polarizers, and dielectrics. Incoming and outgoing wave boundaries and other sources or sinks are not permitted. The EIGENMODE algorithm will find one or several eigenmodes and report the corresponding frequencies.

Keep in mind several factors which can alter the frequencies in subtle ways, especially when comparing the results to analytical models. First, if the geometry is not precisely the same, e.g., due to dimensions snapped to the nearest grid, or stair-stepping, the frequency will be altered. The single most important factor for good agreement is maintaining the same net volume within the simulation. Also, frequencies will always be downshifted slightly because of spatial finite-difference effects. This effect cannot be quantified exactly, but can be estimated as $f_{FD} = f_{true}\,[1-.04(k\delta x)^2]$, where k and δx represent typical wave number and grid spacing within the mode. For well resolved wavelengths, e.g., $k\delta x<<1$, the effect is small.

The EIGENMODE algorithm applies an operator to a given field pattern which grows eigenmodes in the frequency range f_o-δf to f_o+δf, where f_o and δf are the user-specified frequency and frequency window data items, “freq” and “dfreq”. It grows modes as illustrated in the figure below, such that the center frequency, “f_o”, grows the fastest. Each time the operator is applied, the center frequency is grown roughly a factor of 3 above the modes not within the growth window. The operator is applied 30 times, which is sufficient to grow a mode within the frequency window from the noise level to near purity. In general, the computational time required to apply the operator increases as the width of the frequency window is made narrower.

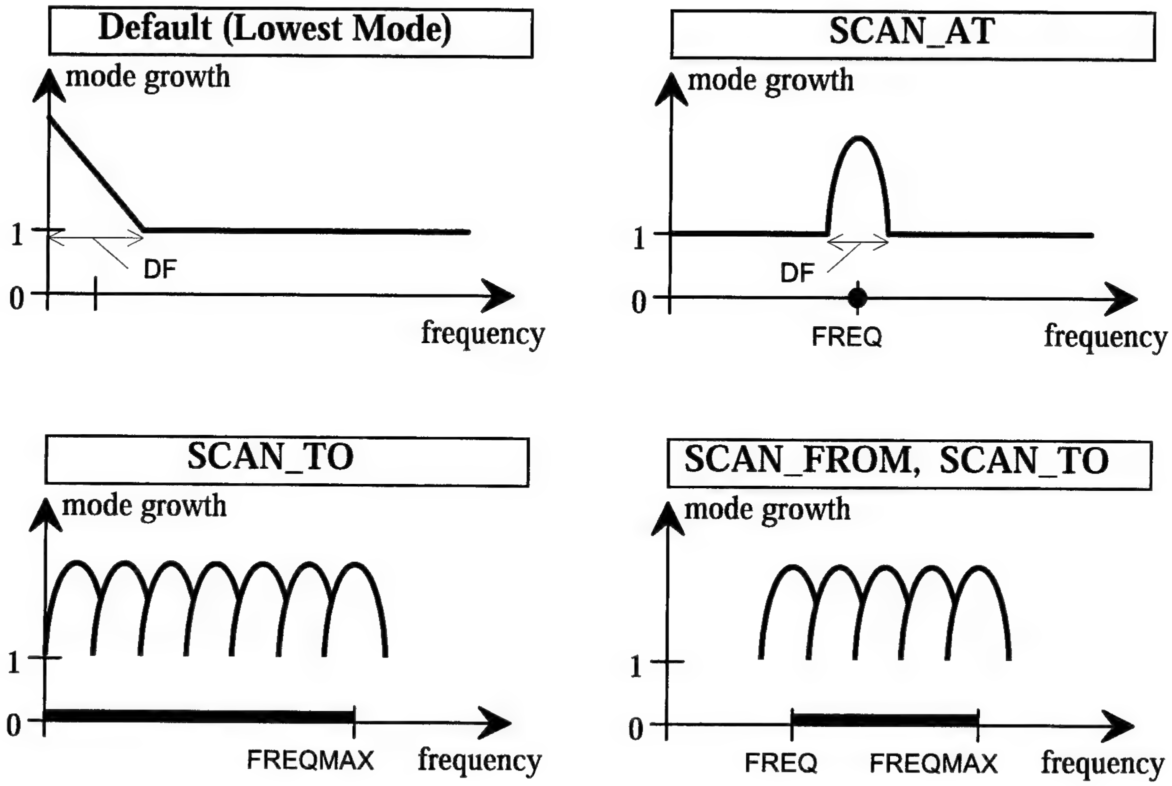


If there are several eigenmodes within the frequency window, e.g., nearly degenerate modes, then the one closest to the center frequency will dominate. A situation where there are two modes within the frequency window is illustrated above. In this case, the algorithm will converge to the nearly central mode, but may contain some contamination from the nearby mode. Thus, a good rule of thumb for the algorithm is that the frequency window should be no larger than the typical spacing between modes. At the end of the run, a concise report will be written to the summary and log files telling the frequency of the converged mode and its confidence level. A poor confidence level is usually an indicator of contamination from a nearly degenerate mode. The near degeneracy can usually be resolved by repeated runs with smaller frequency windows. It is also possible that no modes occur within the search window.

The algorithm contains no inherent preference for the location of the frequency window in the entire spectrum. However, the higher the mode number, the closer together the modes, so that typically df(f) must be made smaller, with the result that more computational time is required. The algorithm will supply a default value for the frequency WINDOW option, based on the overall dimensions of the simulation. This should be satisfactory for most applications, so that typically the WINDOW option need not be used. The exceptions include the presence of a slow-wave structure in the simulation, or when most of the volume of the simulation is metal or outside the region of interest. In such cases, the user will need to supply a more appropriate estimate.

There are typically four ways in which the algorithm can be used to search for eigenmodes. These are:

1) searching for the lowest mode of a system,
2) searching for a mode near some known frequency,
3) searching for all the modes up to some maximum frequency, and
4) searching for all the modes in some known frequency range.



By default, the SCAN_AT frequency is set to zero, which finds the lowest mode of a system. If the frequency WINDOW option needs to be used, df(f) should be set to a conservatively high estimate of what you think the lowest mode is. For a relatively benign geometry, the lowest mode can be found simply by entering the command name, EIGENMODE, with no additional parameters.

The EIGENMODE solver initializes the electric fields with a random field pattern. This provides some field strength in any of the possible eigenmodes. In certain cases, you may wish to seed the field pattern so that only a certain set of modes can be amplified. This may be accomplished by using the NOPRESET option and the PRESET command. The NOPRESET option disables the initialization of electric fields. You then initialize the electric fields with the PRESET command, so that they have the proper modal structure for which you are searching.

The user can choose to search for a mode near some known frequency, instead of the lowest frequency, using the SCAN_AT option. The algorithm will return the single mode closest to this known frequency, or if no modes occur within the search window, it will indicate so in the Eigenmode Report.

Often times the user wishes to find all of the lowest modes. The SCAN_TO option provides this capability. It essentially repeats the SCAN_AT procedure multiple times, moving the search window further up the frequency spectrum each time. The search window is moved by an amount slightly less than the width of the search window, so that no gaps in the spectrum occur. After each search is performed, the resulting mode, if any, is written to the Eigenmode Report. Note that because of the overlap, it is possible for the same frequency to be reported twice in consecutive search windows. The effectiveness of the SCAN_TO option depends on using a suitably small frequency window. Typically, the scan should result in at least every third or fourth window having no eigenmode present. If the scan is reporting a frequency in every window, then the search window is too large and should be decreased. Generally speaking, it is much simpler to run the code for a longer time with a narrower window than it is to attempt to decipher what is happening when there are contaminating modes within the search window.

The user may wish to perform a scan on a restricted part of the spectrum. One common example is a finer detail scan, e.g., one with a smaller window, where two nearly degenerate modes are suspected. The restricted scan is accomplished by using the SCAN_FROM option together with the SCAN_TO option.

The algorithm will automatically start out with a random initial field pattern from which the selected range of eigenmodes can be grown. However, the option exists for the user to initialize the field pattern by using the PRESET command to set the electric fields, E1, E2, and E3. This might be desired in rare circumstances where, for example, there are degenerate modes, and the user wishes to exclude one such mode by starting with a field pattern in which that mode is absent.

The ENERGY_REGION, SAFETY_FACTOR, and ITERATIONS options control features which are normally under algorithm control; hence these options will rarely be used. The ENERGY_REGION dictates the region of the simulation used to calculate the energy quantities, which lie at the heart of determining the eigenmode frequency of a mode. The default is the entire simulation region. The SAFETY_FACTOR controls the estimate of the highest mode in the system, and it is closely related to the Courant limit. By default it is 1.15, e.g., 15% greater than Courant. The ITERATIONS option controls the number of times the eigenmode windowing operator is applied. The default is 30, which should result in a virtually pure mode when there is only one single mode within a search window. By increasing the number of iterations, it is possible to improve the convergence to the center-most mode when near-degeneracy occurs. However, in this situation, it is recommended that a smaller search window be used rather than increasing the number of iterations, since this ultimately provides greater accuracy in a shorter amount of time.

The EIGENMODE algorithm also creates three default Timers that allow the user to observe the field patterns as the algorithm converges to a solution. These timers are named TSYS$EIGENMODE, which triggers when an eigenmode has been found, TSYS$EIGEN, which triggers after each of the 30 iterations, permitting the user to observe the progress of the algorithm, and TSYS$EIGFIRST, which triggers when the algorithm begins a new search window. It may be convenient to use the TSYS$EIGFIRST timer with the GRAPHICS PAUSEOFF command. These timers can be used in RANGE, CONTOUR, and other output commands in the normal manner. Observing the convergence with TSYS$EIGEN can be both mesmerizing and useful, especially if there are closely spaced, competing modes.

In 2D simulations, you can use the MODE option to select the electromagnetic mode, with the choices being TE, TM, or BOTH.

Restrictions:

1. The EIGENMODE can be used only in electromagnetic simulations (no particles).

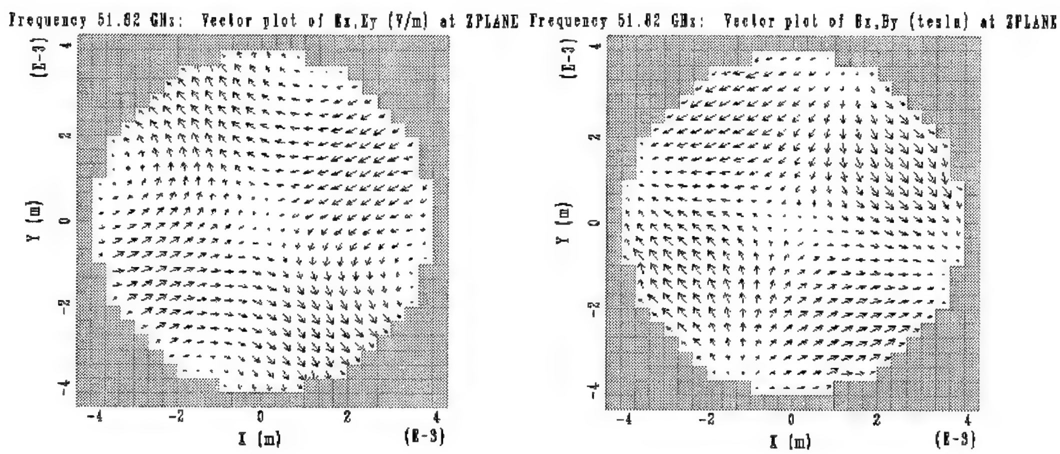
2. No outer boundaries (e.g., PORT) or other possible electromagnetic sources (e.g., DRIVER) or sinks (e.g., CONDUCTANCE) are allowed.

3. To be safe, all regions outside of the simulation region of interest should be made perfectly conducting. (Otherwise, convergence can be adversely affected.)

Example:

The following example is a 3D simulation of a cylindrical pillbox cavity of radius R=4 mm and a height of H= 3.9315 mm. The simulation uses a modest number of cells for resolution. The TE_{211} mode has a frequency of 52.73 GHz. The nearest neighboring mode is the TM_{011} at 47.71 GHz. In this simulation we use the eigenmode SCAN_AT and WINDOW options to look in a narrow frequency band near the TE211 mode. Vector plots of the electric and magnetic field in the cavity are requested in a cross section plane of the pillbox. The cavity eigenmode frequency obtained is 51.82 GHz with a reported error of 0.84%.

```
EIGENMODE SCAN_AT 53GHZ WINDOW 5GHZ ;
VECTOR FIELD E1 E2 ZPLANE TSYS$EIGENMODE ;
VECTOR FIELD B1 B2 ZPLANE TSYS$EIGENMODE ;
```



The figures above show the electric and magnetic field vectors in a cross-section plane of the pillbox.

This page is intentionally left blank.

20. OUTPUT CONTROL

This Chapter covers the following commands:

GRAPHICS
HEADER
DUMP
EXPORT
MOVIE

The GRAPHICS command is used to dispose output to a video screen or a printer, to enable and disable color, to change the orientation from portrait to landscape, to shift the position of the plot on the screen, and to pause and continue plotting. The HEADER command is used to enter background information, such as your personal identification, your organization, etc., which will be reproduced in all graphical output. You can use the DUMP command to write output data to a file for post-processing. You can use the EXPORT command to write data for fields and particles which pass through an outer boundary. (This data can be read in subsequently as an outer boundary for another simulation (IMPORT, Ch. 12).

GRAPHICS Command

Function: Controls graphics disposition and device parameters.

Syntax:

GRAPHICS FILE [filename] [{PS, CGM}] [{LANDSCAPE, PORTRAIT}];
GRAPHICS NOFILE;
GRAPHICS WINDOW [width height [xposition yposition]];
GRAPHICS NOWINDOW;
GRAPHICS COLOR {NORMAL, MONO};
GRAPHICS {FORM, NOFORM};
GRAPHICS PAUSE [duration];
GRAPHICS NOPAUSE;
GRAPHICS {PAUSEON timer, PAUSEOFF timer};

Arguments:

filename	- name of file, user-defined, default = input_file.ps for Postscript and input_file.cgm for CGM.
width, height	- the width and height of the window in pixels.
xposition, yposition	- the position of the top-left corner of the window given as the number of pixels from the top-left corner of the computer screen.
Duration	- the duration of the pause between plots (seconds).
timer	- timer name, defined in TIMER command.

Defaults:

For Windows 95/98/NT
GRAPHICS WINDOW 640 480 -1 -1 ;
GRAPHICS COLOR NORMAL;
GRAPHICS FRAME;
GRAPHICS NOPAUSE;

For Unix
GRAPHICS FILE CGM LANDSCAPE;
GRAPHICS COLOR NORMAL;
GRAPHICS FRAME;
GRAPHICS NOPAUSE;

Description:

The GRAPHICS command provides control over the plotting device and process. Graphics output is directed to two output channels. These are a "window," which provides a run time display of the simulation's progress, and an output graphics file, which may later be viewed or directed to an output device, such as printer.

The FILE option is used to specify the type and layout of the output file to be created. You may specify the file name with the filename argument. The optional arguments, PS and CGM, allow you to specify either a postscript file (PS) or a CGM metafile (CGM). The optional arguments, LANSCAPE and PORTRAIT, specify the orientation of the graphics in the output file. The GRAPHICS FILE command without further options selects the file type as PS for MS-Windows and CGM on Unix. If the file name is not specified, MAGIC assigns a name based on the name of the input file and the output file type, either PS or CGM, referring to a postscript or CGM metafile.

The NOFILE option disables the recording of graphics output in an output file.

The WINDOW option is used to control the size and positioning of the screen window opened when using MS-Windows. In general there is little need to use this option, since on startup MAGIC automatically opens and positions a window for graphics output in the center of the screen. In addition, you can reposition and resize the window with the standard mouse controls once your MAGIC simulation has been started. However, if that is not satisfactory, then the argument's width and height specify the initial window size in pixels. The arguments xposition and yposition specify the top left corner of the window in pixels.

The NOWINDOW option disables the display of the output graphics in a window.

The COLOR options are used to control the color map used for the graphics output. It affects both the output window and the output file. The MONO option produces black-and-white output. The NORMAL option uses a color map with sixteen colors.

The FORM and NOFORM options enable and disable, respectively, the blueprint-style information printed at the bottom of each plot.

The PAUSE / NOPAUSE / PAUSEON / PAUSEOFF options are used to enable/disable a pause between plots. If pause is enabled, you must press the ENTER key on the keyboard after every plot to proceed. PAUSE is ignored with FILE-only graphics. When the duration is specified with PAUSE, the plotting pauses for duration seconds before proceeding with the next plot and does NOT wait for the ENTER key to be pressed.

Restrictions:

1. Only one option may be specified with each GRAPHICS command.

2. The CGM files produced by the Windows version of MAGIC are compatible with a limited number of graphics programs. We, therefore, recommend that users use Postscript files with the Windows version of MAGIC. The Unix CGM files can be viewed or translated into Postscript using the freeware utilities, Gplot or Ralcgm.

3. Screen graphics are currently only available on Windows 95/98/NT and Sun Solaris 2.6 (or later) for SPARC.

Backward Compatibility:

"GRAPHICS SHIFT;" is ignored.

"GRAPHICS NOCOLOR;" is the same as "GRAPHICS COLOR MONO;".

"GRAPHICS PRINTER [filename [type]];" is the same as using the following two commands:

"GRAPHICS FILE [filename] PS;" and "GRAPHICS COLOR MONO;".

"GRAPHICS ORIENTATION" is ignored.

HEADER Command

Function: Allows you to annotate plot output with information that identifies the simulation.

Syntax:

```
HEADER   { ORGANIZATION "company" ,
           AUTHOR "name" ,
           DEVICE "model" ,
           RUN "identifier" ,
           REMARKS "remarks" } ;
```

Arguments:

company - company, organization, or group name.
name - name of individual.
device - name of device.
identifier - run identification.
remarks - remarks.

Description:

The HEADER command allows you to annotate the nature of the particular simulation you are exercising. It may include such information as the name of the author, the organization to which you belong, the name of the device which you are simulating, a run number to identify a particular sequence of simulations, and other information as desired.

The options are as follows. The ORGANIZATION option allows you to enter the name of your company or organization. Your own name should be provided under the AUTHOR option. The DEVICE option allows specification of the generic name or model number of the device being simulated. The RUN option can be used to enter a unique identifier for an individual simulation. Finally, the REMARKS option can be used to record brief comments relevant to the device or study.

Information from the HEADER command is supplied to all modes of output. In plotted output, it is arranged in "blueprint" format on each plot. (Blanks will be substituted for all options not entered.)

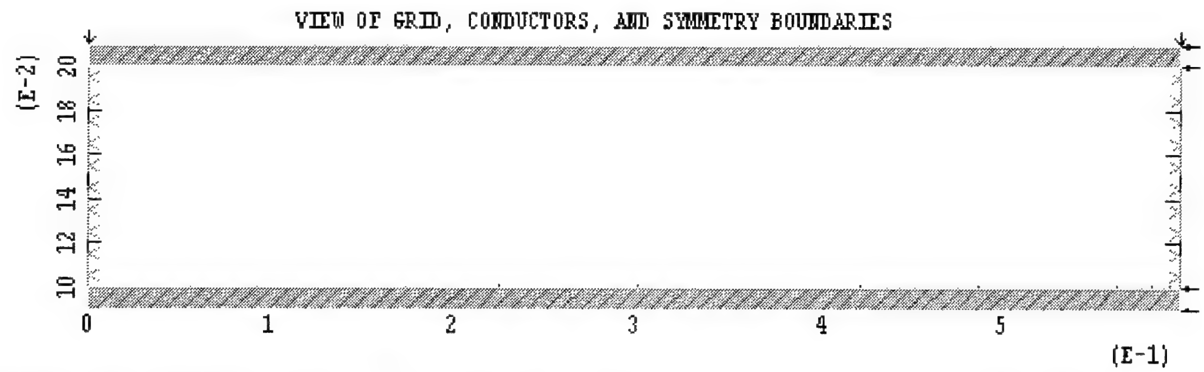
Restrictions:

1. A separate HEADER command is require to enter each option.
2. The quotation marks enclosing the "information strings" are required.
3. Remarks should be brief to avoid truncation in the plotted output.

Examples:

Information from the following HEADER command will be provided in all output.

```
HEADER   ORGANIZATION  "Mission Research Corporation";
HEADER   AUTHOR  "B. Goplen" ;
HEADER   DEVICE  "Cable Exercise" ;
HEADER   RUN  "1" ;
HEADER   REMARKS "Day 2 Seminar" ;
```



Remarks: Day 2 Seminar	Device: Cable Exercise Run No: 1 File: day2.mgc Author: B. Goplen Organization: Mission Research Corporation

MAGIC2D	Version: January 1999	Date: Feb 07,1999	Time: 13:40	Page: 2

Figure shows typical header information in plotted output.

DUMP Command

Function: Controls the DUMP output channel settings.

Syntax: DUMP { TYPE data_type, PREFIX prefix , SUFFIX suffix , FORMAT { ASCII, BINARY } } , ;

Arguments:

data_type	-	type of data (ALL, CONTOUR, OBSERVE, PHASESPACE, RANGE, TABLE, or VECTOR).
prefix	-	prefix to GRD, FLD, PAR and TOC file names.
suffix	-	suffix to GRD, FLD, PAR and TOC file names.

Defaults:

The default prefix is the same as the input file prefix, the default suffix is NONE, and the default FORMAT is ASCII.

Description:

The DUMP output channel writes the results of the output commands in a raw numeric form that can be read by the MAGIC family of codes for post-processing and other simulations.

The TYPE option specifies the data_type. Some data_types are connected with other output commands. In such cases, including CONTOUR, OBSERVE, PHASESPACE, etc., data is intercepted as it is about to be plotted. Only raw data is recorded. For example, the CONTOUR data_type includes field values defined in a plane, but not any information about actual contours. By default, plots will also be produced; however, they can be suppressed by entering NOPLOT.

All data from each data_type is recorded in one of three files: FLD, PAR, and GRD. (The names indicate only the dominant nature of the data.) These three files are opened when the simulation begins, and any empty file is simply deleted at the end of the simulation. A fourth file, TOC (table-of-contents), is also opened to summarize the contents of the other three files. The contents can be reviewed simply by running TOC as an input file. The TOC file (Table of Contents file) contains a list of the data stored in the DUMP files. When the TOC file is used as the input to a MAGIC program, all of the output data from a simulation which is stored in the DUMP files will be replayed in GRAPHICS SCREEN mode. In addition, it is easy to edit the TOC file in order to replay only certain parts of the simulation output, or to redirect the output into a GRAPHICS PRINTER file

You can use the PREFIX and SUFFIX options to create unique names for the four files, as follows:

'prefix'FLD'suffix'
'prefix'PAR'suffix'
'prefix'GRD'suffix'
'prefix'TOC'suffix'

The default prefix is the same as the input file prefix. If the mode is interactive, the default prefix is 'FILE,' and the default suffix is 'NONE.' The file names must conform to requirements of the operating system. On systems using the file extension (such as the PC), a period can be used as the last character of the prefix to make FLD, PAR, GRD, and TOC the file extensions (see Examples, below).

The FORMAT option specifies the data format. The ASCII format is useful for transferring data between computers. The BINARY (default) format is compact and saves disk space. The same format will be used in all three data files. When the ASCII format is used, it is also easy to import the numeric information into spreadsheet programs.

Restrictions:

The full file name constructed from the prefix and suffix must be a valid file name for the operating system in use.

See Also: AUTOGRID, Ch. 11
GRID, Ch. 11
OBSERVE, Ch. 22
FLUX, Ch. 22
RANGE, Ch. 23
CONTOUR, Ch. 24
VECTOR, Ch. 24
PERSPECTIVE, Ch. 24
PHASESPACE, Ch. 24

Examples:

The following set of DUMP commands will cause files to have the names 201.GRD, 201.FLD, 201.PAR, and 201.TOC, which is useful if simulations are given unique numbers. The data will be in binary format in order to save disk space.

```
DUMP PREFIX "201." ;
DUMP FORMAT BINARY ;
DUMP TYPE ALL ;
```

EXPORT Command

Function: Defines data output for of particles and fields passing through a surface.

Syntax: EXPORT { line, area } { POSITIVE, NEGATIVE } file_prefix file_format [timer] ;

Arguments:

line	- name of spatial line, defined in LINE command (2D simulations only).
area	- name of spatial area, defined in AREA command (3D simulations only).
file_prefix	- prefix of file name.
file_format	- file format (ASCII or BINARY).
timer	- name of timer, defined in TIMER command.

Description:

The EXPORT command defines a surface for recording fields and particles for use in an IMPORT command in a subsequent simulation. That is, output from EXPORT provides input for IMPORT.

The EXPORT/IMPORT technique can be effectively used when it is desirable (and reasonable) to break a simulation into two or more parts which can be performed more efficiently. We will use the klystron to illustrate. This technique is desirable in klystrons, because we would like to design cavity N+1 separately, without having to include all of the preceding N cavities in the simulation. It is reasonable, because the wave guides connecting klystron cavities are typically cut off, thus minimizing the backward wave, and because particles in the wave guides all move downstream. Therefore, we would EXPORT a single RF cycle at the outlet of cavity N and IMPORT this cycle over and over again at the inlet of cavity N+1. The location of the surface would be near the midpoint of the drift tube joining the cavities. (Note that EXPORT is not an outer boundary; its presence has no effect on the simulation. The actual outer boundary in the EXPORT simulation might be slightly downstream from the EXPORT surface. By contrast, IMPORT is an outer boundary and therefore part of the simulation perimeter.)

What EXPORT does is write a time record of fields and particles to file(s). You can control only the details of when during the simulation the information is output (e.g., for one full RF cycle after cavity N saturates), but the file contents are determined by the code. In general, it will contain two transverse (in the surface plane) electric field components, but in 2D TM or TE simulations, there will be only one. Only particles crossing the surface in the specified (downstream) direction will be recorded; counter-flowing particles are omitted from the file. The particle coordinates recorded will be relative to the surface, not the coordinate system. Depending on the Lorentz step_multiple, particle records may be written less frequently than field records.

What IMPORT actually does is read this file and apply the fields and particles as an outer boundary. You can control only the details of when during the simulation the file information is applied (e.g., repeating one full RF cycle over and over again). If possible, it is good practice to make the spatial grid near the surface and the electromagnetic time_step and the Lorentz step_multiple the same in the EXPORT and IMPORT simulations. If they are not the same, the IMPORT simulation will redefine the Lorentz time step to match EXPORT and then reset the electromagnetic time_step to provide an integer Lorentz step_multiple (see Restrictions, below). Thus, the particles will always be the same, but the fields may be interpolated in time as well as transverse space.

The EXPORT surface must be conformal with one of the axes and part of the simulation perimeter. In 2D simulations, the surface is represented by a line; in 3D simulations, it is represented by an area, which must be conformal with the X3 coordinate. The sense of the surface is always from outside the perimeter to inside. (This is also the direction of the incoming particles and wave.) If this is in the direction of increasing coordinate, enter POSITIVE; otherwise, enter NEGATIVE.

The file_prefix uniquely identifies field and particle files, i.e., 'file_prefixFLD' and 'file_prefixPAR'. The default file_prefix is the name of the input file. The file_format is either ASCII or BINARY.

The timer specifies the time steps on which the data is exported. If no timer is entered, the default behavior is to export on every time step. If specified, the timer typically will include a contiguous set of time steps. Export will turn on at the end of the first time step indicated in timer. Thus, if timer specifies all time steps from 1000 to 1100, data will actually be exported for steps 1001 through 1100. It is also possible to export data periodically, e.g., from 1000 to 1100 every 10 time steps. In this case, export turns on at the end of time step 1000, and particle data is collected for 10 time steps before being recorded. Thus, the first particle record contains data from steps 1001 to 1010; the next contains particles from 1011 to 1020, etc. If EXPORT is used with a non-unity Lorentz step_multiplier, then the beginning time step should have modulus one, and the ending time step should have modulus zero, relative to the multiple.

Restrictions:

1. Only one EXPORT command is allowed in a simulation. It must follow the DURATION command.

2. EXPORT is not an outer boundary or part of the simulation perimeter (it is an output command).

3. EXPORT data is recorded relative to the surface, not the simulation coordinates.

4. The Lorentz (particle kinematics) time step must be the same in the EXPORT and IMPORT simulations. MAGIC will enforce this by altering it, if necessary. (The electromagnetic time steps need not match. They are subject, however, to the usual integer step_multiple restriction relating Maxwell and Lorentz time steps.)

5. For various reasons, the EXPORT/IMPORT technique works best if there is no backward wave present. (It is typically employed in cut-off geometries, etc.) A backward wave will be trapped in the IMPORT simulation, with possible deleterious results.

See Also: **MAXWELL, Ch. 17**
LORENTZ, Ch. 18
DUMP, Ch. 25
IMPORT, Ch. 12

Examples:

In this 2D simulation, an EXPORT command is used to write both TM and TE components of the electric field as well as particles moving in the positive z-direction through a surface named "Output" to an ASCII file named EXP'icase.'

```
EXPORT Output POSITIVE  EXP'icase' ASCII ;
```

MOVIE Command

Function: Plays MAGIC movie files on a PC.

Syntax: MOVIE PLAY file folder [REPEAT number_cycles] ;

Arguments:

File folder - name of movie folder, e.g. PHASE001.
number_cycles - number of times to replay movie.

Description:

The MOVIE command is used to play MAGIC movie files created during a MAGIC simulation. The keyword PLAY specifies the name of a particular movie folder. The keyword REPEAT is used to set the number of times to cycle through a given movie. The default for this is unity. The data format for movies is a sequence of PCX files stored in individual folders. In addition, because there are large number of PCX files per movie, ensure that the TIMER which is controlling the output command is limited to a few hundred triggering events and ensure that there is sufficient disk space. Any simulation which generates a movie folder will also automatically generate a movie play file. The standard name for this file is the name of the input file with the suffix ".MVY." This file contains all the MOVIE commands required to play the movie frames. Hence, you may view your movies simply by running MAGIC, using it as the input file.

Individual movie folders are created for each output command invoking the MOVIE keyword. A movie folder name consists of a five-character prefix and a three-character numeric. The prefix denotes the output command used for generation of the movie file. The prefix is followed by a three-character sequence number, which indicates the numerical sequence in which a particular command was entered. If, for example, you add the keyword MOVIE to the end of the second PHASESPACE command, MAGIC will record the movie information for the command in a folder named PHASE002. The output commands that allow the MOVIE keyword and their folder prefixes are given in the following table.

Command Name	Folder Prefix
CONTOUR	CONTR
PHASESPACE	PHASE
RANGE	RANGE
VECTOR	VECTR
VIEW3D	3DVUE

Restrictions:

1. Movie files may only be created and played using the PC version of MAGIC.

See Also:

TIMER, Ch. 11
CONTOUR, Ch 24
PHASESPACE, Ch 24
RANGE, Ch 23
VECTOR, Ch 24
VIEW_3D, Ch 24

21. TEXT OUTPUT

This Chapter covers the following commands:

PARAMETER
STATISTICS
TABLE FIELD
TABLE PARTICLES

The commands listed above are designed solely to produce printed output. However, some output commands designed primarily for graphical output also have print options. Such commands are described in other chapters.

The PARAMETER command is used to assign variables which will automatically be recorded in the Summary file. (Otherwise, the variable functions identically to one assigned using an ASSIGN command.) You can use the STATISTICS command to keep track of particle statistics during the simulation. The TABLE command can be used to print tables of electromagnetic field or particle values within a specified spatial object.

PARAMETER Command

Function: Specifies variables to be listed in the SUM file.

Syntax: PARAMETER variable = expression ;

Arguments:

variable	- character variable.
expression	- arithmetic or string expression.

Description:

The PARAMETER command defines variables in the exact same manner as the ASSIGN command. However, in addition to the variable definition, the PARAMETER command also displays the variable in the SUM (summary) file.

The variable type naming conventions and the expression syntax for PARAMETER variables is identical to that for ASSIGN, and the use of PARAMETER variables with other commands is identical to that for ASSIGN commands.

See Also: **ASSIGN, Ch. 6**

Examples:

The following example creates an ASSIGN variable called LARS and a PARAMETER variable called DARS and uses them to create a third variable called HARS.

```
DEFINE LARS = -1 ;            ! not listed in SUM File
PARAMETER DARS = 1. ;         ! listed in SUM File
HARS = LARS + 3*DARS ;        ! result = 2
```

STATISTICS Command

Function: Specifies times at which particle statistics are printed.

Syntax: STATISTICS timer ;

Arguments: timer - timer name, defined in TIMER command.

Description:

The STATISTICS command causes particle statistics to be collected and printed during the simulation. Totals and averages are printed in the LOG file at the end of the simulation. The statistics variables can also be plotted at the user's option (OBSERVE STATISTICS, Ch. 22).

Particle statistics are printed at simulation times specified by the timer. The particle statistics printed are:

number existing at the last measurement,
number created since the last measurement,
number destroyed since the last measurement,
number existing at present, and
error in the number of particles.

The error reveals any variation between the actual count of existing particles and the difference between creation and destruction counts. A nonzero error is indicative of an internal logic error and associated simulation results would be questionable.

At the end of the simulation, several additional statistics are printed:

total number created during the simulation,
total number destroyed during the simulation,
highest number existing during the simulation, and
average number existing during the simulation.

Note that these statistics will be zero in the event that the STATISTICS command has been omitted.

See Also: OBSERVE STATISTICS, Ch. 22

TABLE FIELD Command

Function: Prints a table of values of a specified field in scientific (exponential) notation.

Syntax:

```
TABLE FIELD field object timer
        [ AXES { X1, X2, X3 } { X1, X2, X3 } ]
        [ TRANSFORM function(field, x1, x2, x3) ]
        [ DELIMITER { TAB, COMMA, SEMICOLON, COLON } ]
        [ { NODUMP, DUMP } ]
        [ { PRINT, NOPRINT } ]
        [ FILE file ] ;
```

Arguments:

field	- field component (E1, E2, ...).
object	- name of object, defined in POINT, LINE, AREA, or VOLUME command.
timer	- timer name, defined in TIMER command.
function	- transformation function, defined in FUNCTION command.
file	- name of output file.

Defaults:

The defaults are listed in the following table.

Keyword	Argument	Default Value
AXES	-	X1, X2
TRANSFORM	function	Unity
DELIMITER	-	TAB
{ NODUMP, DUMP }	-	NODUMP
{ PRINT, NOPRINT }	-	PRINT
FILE	file	input_file.LOG

Description:

The TABLE command causes fields associated with the spatial grid to be printed as a table of numbers in scientific notation. The table displays the data more precisely, though less compactly, than a graphical plot.

The object may be a point, a line, an area, or a volume. Only those fields defined within this spatial region will be printed. The TIMER option lets you specify a timer to trigger output of the table.

The AXES option can be used to alter the orientation of the table by specifying which coordinate axis varies horizontally across the printed page and which axis varies vertically. If the spatial_object is three-dimensional, then multiple tables will be produced to include values in the third coordinate. If no axes are entered, the default order for table coordinates is X1 (horizontal), X2 (vertical), and X3 (new page).

Normally, field values are printed in scientific (exponential) notation to six significant digits. This form of highly accurate output is suitable primarily for debugging. However, the TRANSFORM option allows simple transformations, including scaling and formatting, to be performed on the fields, which provides sufficient versatility for virtually any purpose.

The DELIMITER option allows you to specify the delimiter between values printed in the table. The default delimiter is TAB, which results in the appearance of blank spaces between printed values. (This choice allows entire sections of the table to be "copied" from the file and "pasted" into Microsoft Excel for post processing, if desired.)

The DUMP option allows table results to be stored as a data record in the FLD file for post processing. The PRINT option allows table results to be stored as a data record in the LOG file. If desired, this record can be diverted from the LOG file to some other file using the FILE option.

Restrictions:

The total number of TABLE commands in a simulation is limited to 100.

TABLE PARTICLES Command

Function: Prints a table of values for particles in scientific (exponential) notation.

Syntax: TABLE PARTICLES object timer file [{ ASCII, BINARY }] ;

Arguments:

object	- name of object, defined in an AREA command for a 2D simulation and a VOLUME command for a 3D simulation.
timer	- timer name, defined in TIMER command.
function	- transformation function, defined in FUNCTION command.
file	- name of output file.

Defaults:

The default file format is ASCII.

Description:

The TABLE PARTICLES command causes information about the particles inside the specified object to be printed as a table of numbers in scientific notation. The table displays the data more precisely, though less compactly, than a graphical plot. The timer triggers output to the table. Values are printed in scientific (exponential) notation to six significant digits.

The output file begins with the simulation time step, SYS$DTIME, and a table for the species, followed by the particle information. There are no headers or titles in the files. The files only contain the actual particle data. A new line is added for each particle that is incident on the foil or exits from the foil. The data structure of the file is as follows:

```
SYS$DTIME
numSpecies
iSpecies species_name q_over_m m_species
iSpecies species_name q_over_m m_species
...
iSpecies species_name q_over_m m_species
iTimeStep iSpecies charge x1 x2 x3 p1 p2 p3
iTimeStep iSpecies charge x1 x2 x3 p1 p2 p3
iTimeStep iSpecies charge x1 x2 x3 p1 p2 p3
...
iTimeStep iSpecies charge x1 x2 x3 p1 p2 p3
```

For the ASCII data option, the Fortran-style formats are:

```
Sys$dtime: e13.6
NumSpecies: i4
species table: (i4,1x,a32,1x,e13.6,1x,e13.6)
particle table: (i9,1x,i4,7(1x,e13.6))
```

For BINARY, all values are four bytes long except the species name, which is a character value with a length of 32 bytes. There are no delimiters in the BINARY format.

22. TIME PLOTS

This Chapter covers the following commands:

(2D and 3D simulations)
OBSERVE [options]
OBSERVE SET_OPTION
OBSERVE FIELD
OBSERVE FIELD_ENERGY
OBSERVE FIELD_INTEGRAL
OBSERVE FIELD_POWER
OBSERVE PARTICLE_STATISTICS
OBSERVE RESONANT_PORT
OBSERVE STRUT

(2D simulations only)
OBSERVE CIRCUIT
OBSERVE TRAMLINE

(3D simulations only)
OBSERVE CREATED
OBSERVE DESTROYED

You use the OBSERVE commands to plot the value of a simulation variable vs. time. There are many processing options which can be applied to the data, and these are described by the OBSERVE [options] command. Defaults are set for many of these options, which can be reset using the OBSERVE SET_DEFAULT command. All of the remaining OBSERVE commands involve plotting a particular variable_type, such as FIELD, FIELD_INTEGRAL, FIELD_ENERGY, etc. A few measurement_types may apply only to 2D or 3D simulations.

OBSERVE [options] Command

Function: Specifies data-processing options for OBSERVE commands.

Syntax:

```
OBSERVE [ variable_type ] [ arguments ]
        [ SUFFIX suffix ]
        [ TRANSFORM function(y,t) ]
        [ { DIFFERENTIATE, INTEGRATE } ]
        [ FILTER {LO_PASS, STEP} t_filter]
        [ AMPLITUDE frequency cycles ]
        [ FFT { MAGNITUDE, COMPLEX } ]
        [ WINDOW TIME minimum maximum ]
        [ WINDOW FREQUENCY minimum maximum ]
        [ { PLOT, NOPLOT } ]
        [ { NODUMP, DUMP } ]
        [ { NOPRINT, PRINT } ] ;
```

Arguments:

variable_type	-	type of simulation variable, see other OBSERVE commands. The variable types FIELD, FIELD_ENERGY, FIELD_INTEGRAL, FIELD_POWER, PARTICLE_STATISTICS, SET_OPTION, and STRUT are available in both 2D and 3D simulations. The options CIRCUIT and TRAMLINE are available only in 2D simulations. The options CREATED and DESTROYED are available only for 3D simulations.
arguments	-	differs for each variable_type; see other OBSERVE commands.
suffix	-	name given to associated system variable, OBS$suffix.
function	-	transformation function, defined in FUNCTION command.
y	-	dummy argument representing the variable.
t	-	dummy argument representing simulation time.
t_filter	-	filter time constant (sec).
frequency	-	reference frequency for amplitude analysis (Hz).
cycles	-	number of periods in amplitude-analysis time window.
minimum, ...	-	boundaries of FFT integration (sec) and plot (Hz).

Defaults:

The default values for the data processing options are listed below. Note that these default values can be reset with OBSERVE SET_DEFAULT, Ch. 22. The default operations are automatically applied to each OBSERVE command when the specific option is not named explicitly in the command. For example, if you change the FFT default to MAGNITUDE, using OBSERVE SET_DEFAULT, all subsequent OBSERVE commands will automatically trigger an FFT operation, even if the FFT option is not entered in the subsequent commands.

Keyword	Argument	Default
SUFFIX	suffix	1, 2, 3, ...
TRANSFORM	function	unity (no transformation)
INTEGRATE	n.a.	(integration is not performed)
DIFFERENTIATE	n.a.	(differentiation is not performed)
FILTER	tfilter	(no filter)
AMPLITUDE	frequency	0 (no analysis is performed)
AMPLITUDE	cycles	0 (no analysis is performed)
WINDOW TIME	minimum, maximum	entire range
WINDOW FREQUENCY	minimum, maximum	entire range
{ PLOT, NOPLOT }	n.a.	PLOT (output is plotted)
{ NODUMP, DUMP }	n.a.	NODUMP (output is not dumped)
{ NOPRINT, PRINT }	n.a	NOPRINT (output is not printed)

Description:

The OBSERVE command provides the capability to output a time plot, i.e., a simulation variable as a function of time. Individual OBSERVE commands are structured for each variable_type (e.g., OBSERVE FIELD for field variables, etc.), since each requires different arguments. Some variable_types apply only to 2D or to 3D, while others apply to both 2D and 3D. The individual OBSERVE commands which describe each variable_type and their unique arguments follow in this Chapter. A special case occurs when no variable_type or arguments are specified. In this case, the TRANSFORM option must be used to calculate and display an analytic function of the time variable, user-defined variables, system variables, or other observe variables. This section, the OBSERVE [options] command, describes the general data processing operations, including the TRANSFORM option, (e.g., FFT) which can be entered in any OBSERVE command.

Associated with each OBSERVE command is a system variable, OBS$suffix, where the suffix can be specified with the SUFFIX option. If the SUFFIX option is not employed, then the default suffix will be a number, e.g., OBS$1, OBS$2, OBS$3, etc., corresponding to the order in which the OBSERVE commands are entered. This system variable can be used in the same manner as a variable created using the ASSIGN command. Its value is updated at every observation interval. Use of this variable in a FUNCTION command can provide feedback to an incoming wave (PORT, Ch. 12), a current-density source (DRIVER, Ch. 19), or a voltage source (CIRCUIT, Ch. 19). Its use in another OBSERVE TRANSFORM function allows a single observation involving multiple variables, e.g., to obtain a time plot of impedance from voltage and current observations. Its use in a PARAMETER command after the simulation, i.e., between the START and STOP commands, can provide a way to spot-check simulation behavior from the SUM file.

There are six possible data processing operations: TRANSFORM, DIFFERENTIATE, INTEGRATE, FILTER, AMPLITUDE, and FFT. (If you specify more than one operation on the data, please be aware that the operations are always performed in this order, and not in the order you place them in the command.) Each operation causes the variable data to be modified in a particular fashion. Except for the FFT operation, the modified data is passed to the OBS$suffix system variable, as described above.

The TRANSFORM option is used to apply a transformation to the variable. The function has two dummy arguments, y and t. Other variables and functions, including other observed variables, may be referenced by the transformation function in the normal fashion. (See the FUNCTION command for details. Note that when

another observed variable is used in a transformation, its value is obtained from the previous time step, and not the current time step.) The name of the transformation function will appear on the y-axis of the plot.

The DIFFERENTIATE option is used to differentiate the variable with respect to time. Similarly, the INTEGRATE option integrates the variable with respect to time. These two options are mutually exclusive; you may specify one or the other, but not both.

The FILTER option is used to apply either a LO_PASS or STEP filter. The LO_PASS filter acts as an RC filter to the variable. It results in a new signal, g_n, at time step n, given by the following formula:

$$g_n = f s_n + (1-f)\, g_{n-1},$$

where s_n is the original observed variable at time step n, and f is the filter fraction, a number between zero and one. It is assumed that $g_0 = 0$. The above formula is a discrete approximation of the traditional RC-type filter given by:

$$g(t) = \frac{1}{t_{\text{filter}}} \int_0^t dt' s(t') e^{-(t-t')/t_{\text{filter}}},$$

where the filter time constant controls the time scale over which the filter is applied and corresponds to t_{filter} = 1/RC of an RC circuit.

The alternate STEP filter provides time averaging of the original observed variable over the time period given by t_{filter}, as described by the formula

$$g(t) = \frac{1}{t_{\text{filter}}} \int_{t-t_{\text{filter}}}^t dt' s(t').$$

STEP filtering is more effective on signals with known periodic behavior. You should set t_filter to an integer number of periods to get a smooth average over the periodic phenomenon. A common example of the use of a STEP filter is when net Poynting power is desired, since the instantaneous Poynting power, without filtering, contains an undesired oscillation at twice the frequency of a wave. If a LO_PASS filter were used instead of STEP, some visible remnant of the oscillation would still be present after filtering.

The AMPLITUDE option is used to obtain an amplitude envelope of an oscillating signal, as well as a precise frequency and Q value (associated with loss). The analysis uses a simple peak-finding procedure driven by a regular sample clock, preceded by a digital filter to remove noise. You must specify a reference frequency, and the width of the time window, in cycles, which roughly specifies the impulse response duration of the digital filter. This analysis can be applied to observe data which contains a fairly regular oscillating signal and works best if the specified reference frequency is within 50% of the actual signal frequency. Four plots are generated. These plots are:

a) the observe data itself,
b) amplitude vs. time,
c) precise frequency vs. time, and
d) -(2/ω)*(*d*/*dt*) log(amplitude), e.g., $1/Q$, vs. time.

The third plot will return the precise frequency of the signal to great precision, typically to 0.001% or better with low-noise signals. This information can be valuable when tuning high-Q cavities, especially when beam or other loading effects modify a cavity's operating frequency from its ideal eigenmode frequency. The fourth plot shows the exponential decay parameter, $1/Q$, based on an assumed amplitude decay going as $e^{-\omega t/(2Q)}$. (The inverse of Q is plotted to circumvent the singularity which occurs for an undamped signal where Q goes to infinity.) Zero will be plotted whenever the amplitude is increasing instead of decaying. Generally speaking, the results are insensitive to the width of the time window, and a value of ncycles of 4 or 8 is recommended. The amplitude analysis is impervious to signal offset as well as harmonic content; however, it can be damaged by noise. The primary means of dealing with noisy data is to increase the width of the time-window, ncycles, to 8, 16, or even greater values. See the example at the end of this section for an illustration of its use.

The FFT option is used to specify a Fourier transform. You can specify a plot of either the magnitude or the complex parts of the FFT. The WINDOW TIME option can be used to specify the time interval, from minimum to maximum, used for the Fourier integration. Similarly, the WINDOW FREQUENCY option can be used to specify the frequency boundaries for plots of the transformed data.

The actual FFT calculation proceeds as follows. First, the variable between minimum and maximum time is linearly interpolated onto a new set of N uniformly spaced points, such that $N=2^M$ is the next largest exact power of two. The exact interpolation times are $t_j=t_{lo}+(j-{}^1/_2)\mathrm{D}t$, where $\mathrm{D}t=(t_{hi}-t_{lo})/N$. The FFT frequencies are $f_k=k\mathrm{D}f$, where $\mathrm{D}f=(t_{hi}-t_{lo})^{-1}$, and run from k=0 to k=N/2, e.g., $f_{max}=(2\mathrm{d}t)^{-1}$. Starting from the N interpolated variable values, S_j, the FFT is given by:

$$\tilde{S}_k = \sigma_k \Delta t \sum_{j=1,N} S_j \, e^{-i2\pi f_k t_j} \quad \text{, e.g.,} \quad \tilde{S}(f) = 2\int_{t_{lo}}^{t_{hi}} dt\, S(t)\, e^{-i2\pi f t} \quad ,$$

where σ_k=1 for k=0 and σ_k=2 otherwise. It is precisely the quantity $\tilde{S}_k$ which is displayed in the FFT plots. The inverse formula for this convention is:

$$S_j = \Delta f \sum_{k=0,N/2} \mathrm{Re}\left\{\tilde{S}_k \, e^{i2\pi f_k t_j}\right\} \quad \text{, e.g.,} \quad S(t) = \int_0^{f_{\max}} df\, \mathrm{Re}\left\{\tilde{S}(f)\, e^{i2\pi f t}\right\} \quad .$$

The normal factor of 2π is missing because of the integration over frequency, f, instead of angular frequency, ω. The extra factor of two in the transform is compensated for by an integration over only positive frequencies in the inverse transform, a common manipulation where pure, real signals are concerned.

The PLOT, DUMP, and PRINT options allow time plot results to be stored and displayed by three different methods:

1) as a graphical plot in a file or on screen,
2) as a data record in the GRD file, or
3) as printed column data in the LOG file.

A graphical plot in a file or on screen is the normal result of an OBSERVE command. Under certain circumstances, it may be desirable to have particular time plots displayed, but not stored, to save disk space, or perhaps stored, but not displayed, for some other reason. The PLOT, NOPLOT, DUMP, and NODUMP options can be applied to individual time plots; i.e., it is not necessary to treat all the time plots in the same way.

Alternatively, DUMP TYPE OBSERVE or DUMP TYPE ALL commands can be used to record all time plots in the GRD file (DUMP, Ch. 25).

The PRINT option can be used to generate a table of time plot data in the LOG file after the last time step has executed. This option cannot be applied separately to individual time plots. Therefore, the last PRINT or NOPRINT option entered will govern printing to the LOG file.

Normally, time plot data is stored and displayed only once (after the last time step has been executed). On certain platforms, it is possible to display and store time plot data at any time during the simulation. On the PC, simply depress the F5 key on the keyboard (KEYBOARD, Ch. 9).

During a run, the time-plot data is stored in the binary file, "filename.OBS". In the event of an unexpected program termination or power outage, it is sometimes possible to retrieve this data with the POSTER program.

See Also:

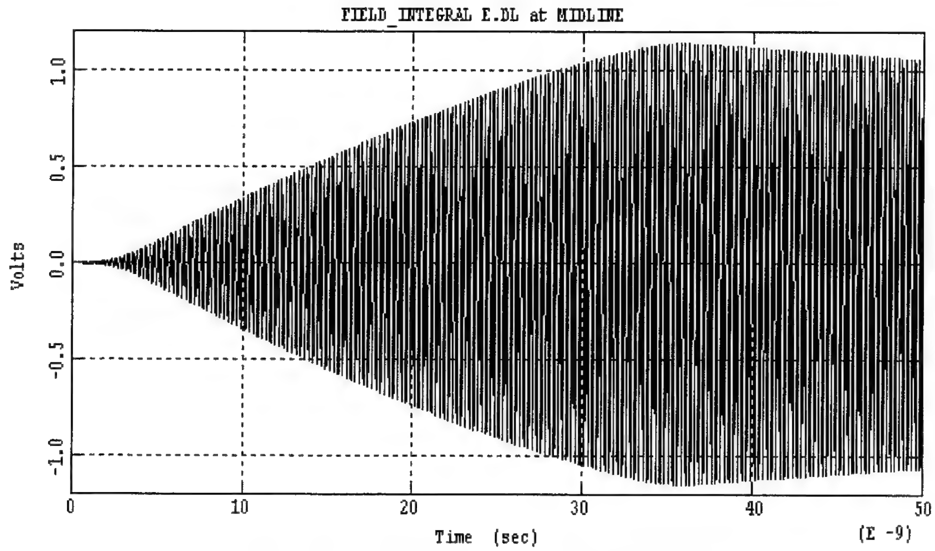
ASSIGN, Ch. 6
FUNCTION, Ch. 6
KEYBOARD, Ch. 9
DUMP, Ch. 25
OBSERVE CIRCUIT, Ch. 22
OBSERVE CREATED, Ch. 22
OBSERVE DESTROYED, Ch. 22
OBSERVE FIELD, Ch. 22
OBSERVE FIELD_ENERGY, Ch. 22
OBSERVE FIELD_INTEGRAL, Ch. 22
OBSERVE FIELD_INTEGRAL, Ch. 22
OBSERVE SET_DEFAULT, Ch. 22
OBSERVE STRUT, Ch. 22
OBSERVE TRAMLINE, Ch. 22

Example:

The following four figures illustrate the use of the AMPLITUDE diagnostic. The simulation is for a cubical cavity with resonant frequency of 4.0 GHz excited by a linear current source from the top to bottom along the midline. (A line was defined, called "midline," in order to observe the voltage between the top and bottom walls of the cavity.) The material was specified as NICHROME so that the losses could be easily observed. The drive was turned on and off smoothly. The observe command that produced these figures was:

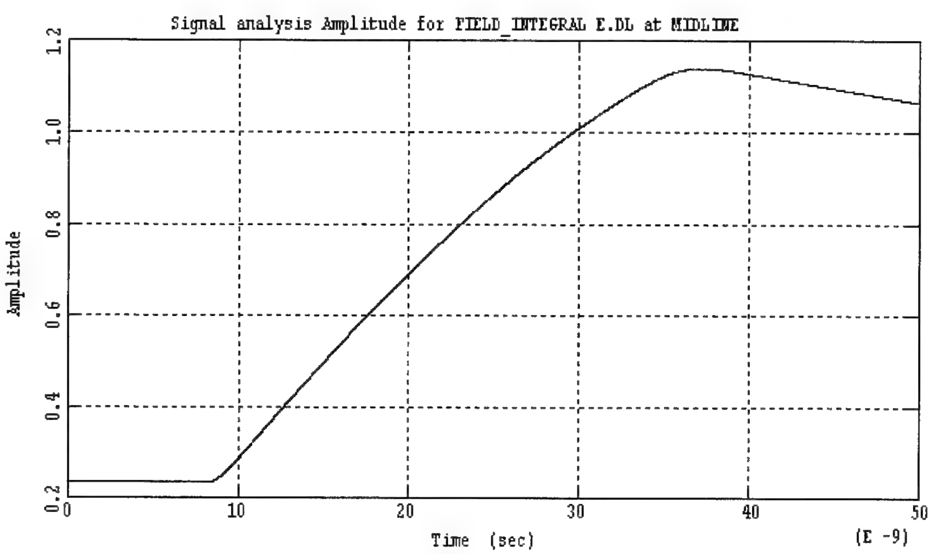
```
OBSERVE FIELD_INTEGRAL E.DL midline AMPLITUDE 4.0GHz 8 ;
```

The figure below shows the signal from the cavity. The amplitude increases while the drive is on and then begins to decay quickly because of the high losses from the nichrome wall material.



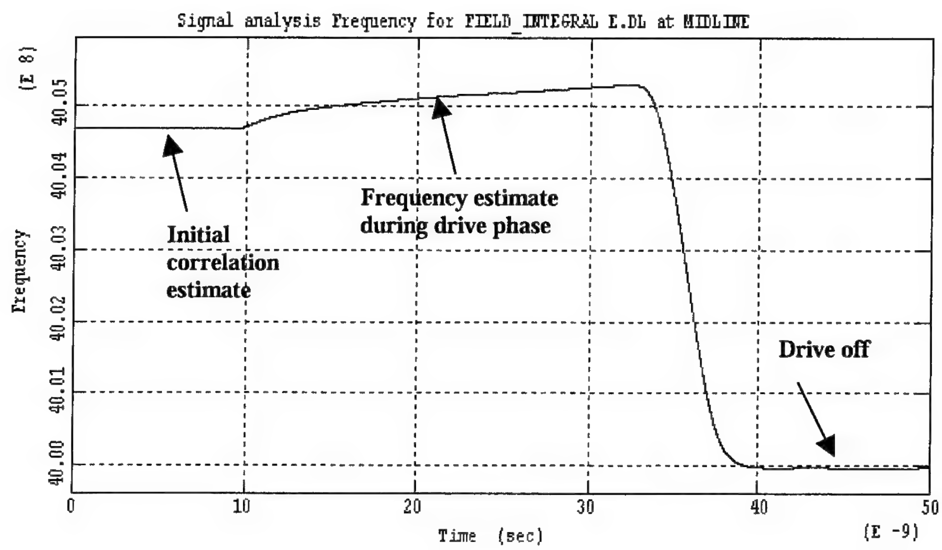
Cavity voltage along midline versus time.

The amplitude of the signal shown in the figure above begins to decay after the drive signal is turned off. The figure below shows the plot for amplitude versus time produced by the diagnostic. This is essentially the positive envelope of the signal. Note that the amplitude displayed is constant at about 0.23 for times below 9 nanoseconds. This initial delay in the display is deliberate and is intended to prevent misinterpretation of possibly inaccurate early-time data due to the initial response time of the digital filter.



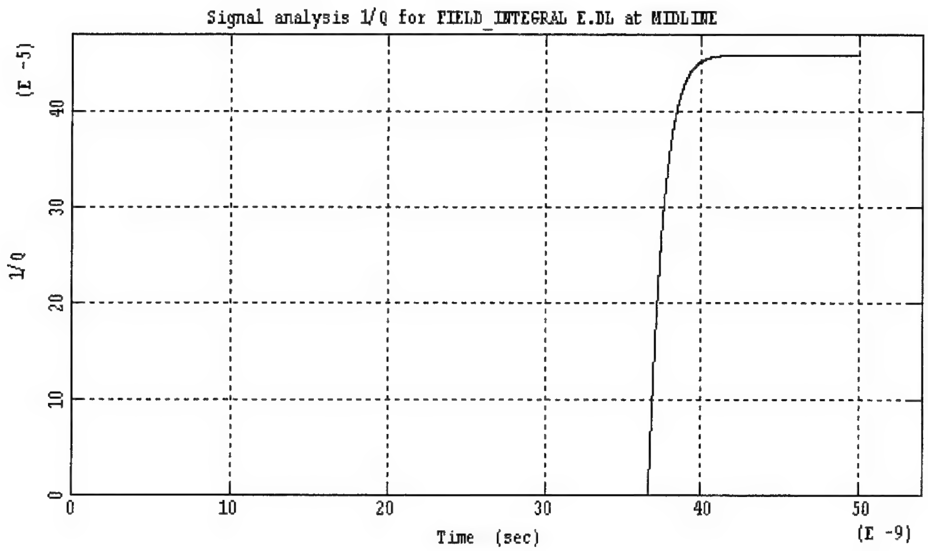
Amplitude envelope of the signal in the preceding figure.

The following figure shows the frequency vs. time. It reveals a decrease in the frequency of the observed voltage after the drive signal is turned off. The frequency of the drive is 4.01 GHz, so that the cavity resonates at 4.005 GHz while the drive is on. (This is the mean value of the drive frequency and cavity resonant frequency, which is the expected early-time frequency of a driven resonant oscillator with damping.) The slight stair stepping or "digital ripple" in the frequency plot after turning off the drive has an rms amplitude of less than 0.0001%. Frequency traces for clean signals are often this accurate, and round-off error can cause a "digital ripple" in the plot that is simply fluctuation of the least significant bits. This frequency estimator works by first maximizing (vs. frequency) a windowed correlation with sine and cosine functions over roughly the period specified by NCYCLES. This initial frequency estimate is then used to set the digital filter center frequency and the rate of a digital phase locked loop (DPLL), which used in conjunction with a level crossing detector, provides the real-time frequency data. The DPLL further reduces jitter in the frequency vs. time estimate, and the user-supplied value of NCYCLES is also used to set the DPLL time constant.



Real-time frequency display.

The following figure shows the 1/Q analysis. When the signal is not decaying (while the drive is on), zero is plotted for 1/Q instead of negative values. As the signal begins to decay, the 1/Q value then settles to about 4.6E-4, the value for the nichrome cubical cavity at 4.0 GHz.



Real-time 1/Q display.

OBSERVE SET_OPTION Command

Function: Resets default options for all subsequent OBSERVE commands.

Syntax: OBSERVE SET_OPTION [options] ;

Arguments: options - data-processing options (OBSERVE [options], Ch. 22).

Defaults:

The default values for the data processing options are listed elsewhere (OBSERVE [options], Ch. 22). Note that these default values can be reset repeatedly (OBSERVE SET_OPTION, Ch. 22). The <u>default operations are automatically applied</u> to OBSERVE commands, even though the specific operation is not named explicitly in the command. For example, if you reset the FFT default from NONE to MAGNITUDE, all subsequent OBSERVE commands will automatically trigger an FFT operation, even if the FFT option is not entered in the subsequent commands.

Description:

The command, OBSERVE SET_OPTION, can be used to reset the OBSERVE option default values, thus saving the user from having to enter options in each OBSERVE command explicitly. The new defaults will apply to all subsequent OBSERVE commands until another OBSERVE SET_OPTION command is entered.

One common usage of this command will be to set the default INTERVAL step_multiple for all time plots. Most of the data processing options can have their defaults reset, with the sole exceptions being the TRANSFORM and NAME options.

See Also: **OBSERVE [options], Ch. 22**

Examples:

In this example, the default step_multiple is reset for all measurements, and the default filter fraction is reset for three measurements. Note that the default filter is applied even though the FILTER option is not explicitly included in the OBSERVE FIELD commands.

```
OBSERVE  SET_OPTION  INTERVAL  10 ;
OBSERVE  FIELD E1 Point_A ;
OBSERVE  FIELD B3 Point_A ;
OBSERVE  SET_OPTION FILTER 0.01 ;      ! ... filter by default
OBSERVE  FIELD E3 Point_A ;
OBSERVE  FIELD B1 Point_A ;
OBSERVE  FIELD B2 Point_A ;
OBSERVE  SET_OPTION FILTER 1.0 ;       ! ... back to no-filter
OBSERVE  FIELD E2 Point_A ;
```

OBSERVE CIRCUIT Command

Function: Specifies circuit variable to be plotted vs. time in a 2D simulation.

Syntax:

OBSERVE CIRCUIT poisson variable
[options] ;

Arguments:

poisson	-	name of Poisson solution, defined in POISSON command.
variable	-	circuit variable (see CIRCUIT, Ch. 12). = CHARGE, CURRENT, VOLTAGE, ENERGY, POWER, DCHARGE, DVOLTAGE, DENERGY.
options	-	processing options (see OBSERVE [options], Ch. 22).

Description:

The command, OBSERVE CIRCUIT, is used to specify a circuit model variable to plot vs. time. The poisson name specifies the circuit. (See CIRCUIT, Ch. 12 a list of the valid observable variables.)

See Also:

CIRCUIT, Ch. 12
POISSON, Ch. 19
OBSERVE [options], Ch. 22

Restrictions:

This command is available only in 2D simulations.

Examples:

The following commands instruct the code to observe and record the circuit variables, CURRENT and VOLTAGE. At the end of the simulation, plots of the time histories of these variables are produced. The observe variables are recorded every time step, since the step_multiple is left at its default value.

```
OBSERVE CIRCUIT MYTEST CURRENT ;
OBSERVE CIRCUIT MYTEST VOLTAGE ;
```

OBSERVE CREATED Command

Function: Specifies particle creation variable to be plotted vs. time (3D simulations only).

Syntax: OBSERVE CREATED object species variable [options] ;

Arguments:

object - name of conducting spatial object (ALL, or specific object defined in VOLUME command).

species - name of particle species (ALL, ELECTRON, PROTON, or defined in SPECIES command).

variable - particle creation variable (see list below).
CHARGE - cumulative charge (Coul).
CURRENT - instantaneous current (A).
ENERGY - cumulative energy (joules).
POWER - instantaneous power (watts).
VOLTAGE - power/current (eV).

options - data processing options (see OBSERVE [options], Ch. 22).

Description:

The OBSERVE CREATED command is used to specify a particle creation variable to be plotted as a function of time. It applies only to particles created on a specified, conducting spatial object. This command applies only to 3D simulations.

The arguments are defined as follows. First, the spatial object is specified. Normally, this will be the name of a single conducting object. However, you can include all conducting objects by entering ALL. Next, enter the particle species, again using ALL if all species are to be included. Finally, enter the particle creation variable as listed above.

Restrictions:

1. This command is available only in 3D simulations.
2. The spatial object must be conducting.
3. The OBSERVE step_multiple must be an integer multiple of the EMISSION step_multiple to avoid discontinuous results for the POWER and CURRENT variables.

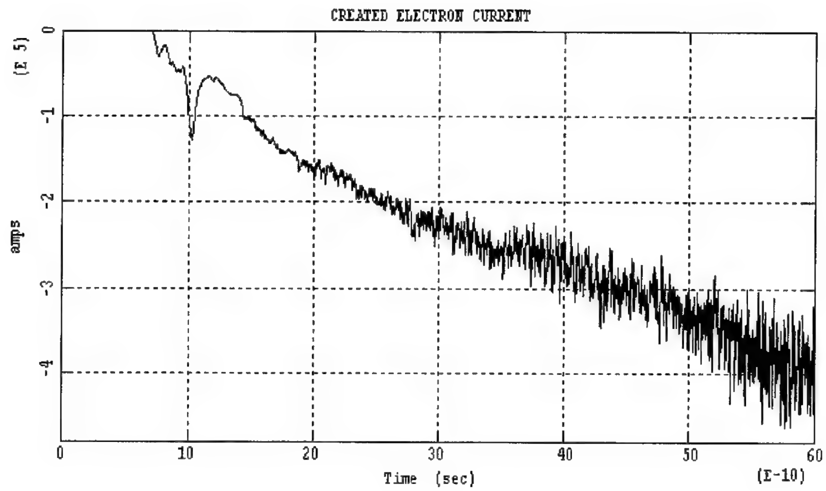
See Also:

VOLUME, Ch. 10
CONDUCTOR, Ch. 14
OBSERVE [options], Ch. 22

Examples:

The following figure was produced by the following command in a 3D simulation of a MITL.

```
OBSERVE CREATED CATHODE ELECTRON CURRENT ;
```



This illustrates instantaneous created electron current on the surface of the conductor cathode.

OBSERVE DESTROYED Command

Function: Specifies particle destruction variable to be plotted vs. time (3D simulations only).

Syntax: OBSERVE DESTROYED object species variable [options] ;

Arguments:

object - name of conducting spatial object (ALL, or specific object defined in VOLUME command).

species - name of particle species (ALL, ELECTRON, PROTON, or defined in SPECIES command).

variable - particle destruction variable (see list below).
CHARGE - cumulative charge (Coul).
CURRENT - instantaneous current (A).
ENERGY - cumulative energy (joules).
POWER - instantaneous power (watts).
VOLTAGE - power/current (eV).

options - data-processing options (see OBSERVE [options], Ch. 22).

Description:

The OBSERVE DESTROYED command is used to specify a particle destruction variable to be plotted as a function of time. It applies only to particles destroyed on a specified, conducting spatial object. This command applies only to 3D simulations.

The arguments are defined as follows. First, the spatial object is specified. Normally, this will be the name of a single conducting object. However, you can include all conducting objects by entering ALL. Next, enter the particle species, again using ALL if all species are to be included. Finally, enter the particle destruction variable as listed above.

Restrictions:

1. This command is available only in 3D simulations.

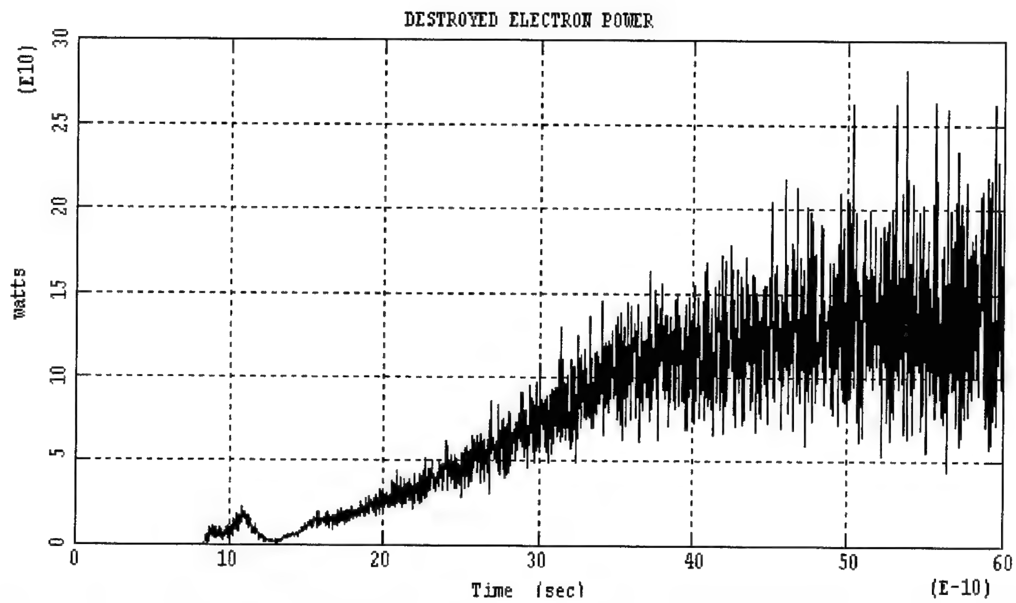
2. The spatial object must be conducting.

See Also: **VOLUME, Ch. 10**
CONDUCTOR, Ch. 14
OBSERVE [options], Ch. 22

Examples:

The following figure was produced by the following command in a 3D simulation of a MITL.

```
OBSERVE DESTROYED ANODE ELECTRON POWER ;
```



This figure shows the instantaneous power deposition of electrons intercepted by the conductor anode.

OBSERVE FIELD Command

Function: Specifies electromagnetic field variable to be plotted vs. time.

Syntax: OBSERVE FIELD field object [options] ;

Arguments:

field - field component (see list below).
E1, E2, E3, B1, B2, B3, J1, J2, J3, SIGMA, Q0,
E1ST, E2ST, E3ST, B1ST, B2ST, B3ST,
E1AV, E2AV, E3AV, B1AV, B2AV, B3AV,
D1, D2, D3, DIE1, DIE2, DIE3, QERR, (this line, 2D only).

object - name of spatial object, defined in POINT, LINE, AREA, or VOLUME command.

options - data-processing options (OBSERVE [options], Ch. 22).

Description:

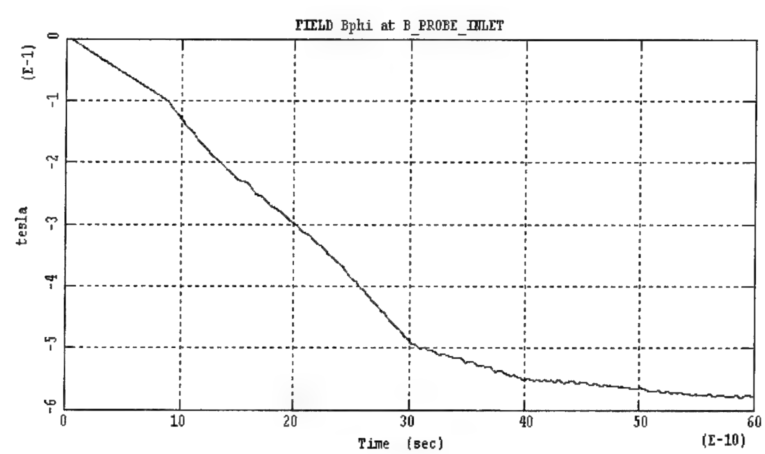
The OBSERVE FIELD command is used to specify a field at a spatial object to be plotted vs. time. If the specified object is a point, the field at that location will be measured. For any spatial object other than a point, an average value of the field over the object is calculated. For example, if the object is a volume, then the field will be integrated over the volume, and the result will be divided by the total volume to give an average value of the field.

See Also: **MAXWELL algorithms, Ch. 22**
OBSERVE [options], Ch. 22

Examples:

The field, B2, is measured at point, "PROBE_INLET," and a time history plot is produced.

```
POINT   PROBE_INLET   0CM,8CM,1PI ;
OBSERVE  FIELD  B2  PROBE_INLET  ;
```



This figure shows a time history of the magnetic field at the location B_PROBE_INLET.

OBSERVE FIELD_ENERGY Command

Function: Specifies electromagnetic field-energy variable to be plotted vs. time.

Syntax: OBSERVE FIELD_ENERGY variable object [options] ;

Arguments:

variable	- energy variable (see list below). E1, E2, E3, B1, B2, B3, EM, ELECTRIC, MAGNETIC, TM, TE (this line, 2D only).
object	- name of spatial object, defined in AREA command (2D) or VOLUME command (3D).
options	- data processing options (see OBSERVE [options], Ch. 22).

Description:

The OBSERVE FIELD_ENERGY command is used to specify an electromagnetic field-energy variable to be plotted vs. time.

The variable specification allows various combinations of fields to be included. For example, the contribution from any individual field component can be isolated by specifying E1, E2, E3, B1, B2, or B3. The total electromagnetic field energy is obtained by entering EM, while the electric and magnetic components are obtained using ELECTRIC and MAGNETIC. For 2D simulations only, entering TM yields the total electromagnetic energy in the TM mode (E1, E2, B3). Similarly, entering TE yields the total electromagnetic energy in the TE mode (B1, B2, E3). The spatial object is used to define boundaries which limit the spatial extent of the energy calculation. Normally, the object will include the entire simulation, but you may wish to restrict the measurement to some local region.

See Also: **OBSERVE [options], Ch. 22**

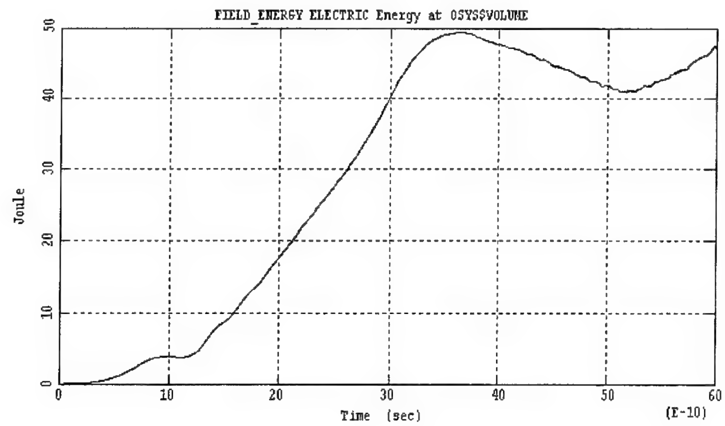
Restrictions:

The spatial object must be an area for 2D simulations and a volume for 3D simulations.

Example:

In a 3D simulation of a coaxial MITL, we can obtain the electric field energy using the following command line.

```
OBSERVE FIELD_ENERGY ELECTRIC OBS$VOLUME ;
```



This figure shows the time history of the electric field energy in the simulation volume OSYS$VOLUME.

OBSERVE FIELD_INTEGRAL Command

Function: Specifies electromagnetic field integral to be plotted vs. time.

Syntax: OBSERVE FIELD_INTEGRAL { E.DL, H.DL, J.DA, Q.DV } object [options] ;

Arguments:
object - spatial object defined in the POINT, LINE, AREA, or VOLUME commands.
options - data-processing options (OBSERVE [options], Ch. 22).

Description:

The OBSERVE FIELD_INTEGRAL command is used to specify a field integral over a spatial object to be plotted vs. time. The integral will carry the combined units of the field and the spatial differential, DL (meters), DA (meters squared), or DV (meters cubed). Thus, E.DL has units of volts, H.DL has units of amps, J.DA has units of amps, and Q.DV has units of coulombs. The object specified must be consistent with the type of integral that is specified. That is, E.DL and H.DL require a POINT or LINE object in 2D and a LINE object in 3D, J.DA requires either an AREA or a LINE in 2D and an AREA in 3D, and Q.DV requires an AREA in 2D and a VOLUME in 3D. If desired, the integral result can be transformed using OBSERVE [options] (See Examples, below).

See Also: **MAXWELL algorithms, Ch. 17**
OBSERVE [options], Ch. 22

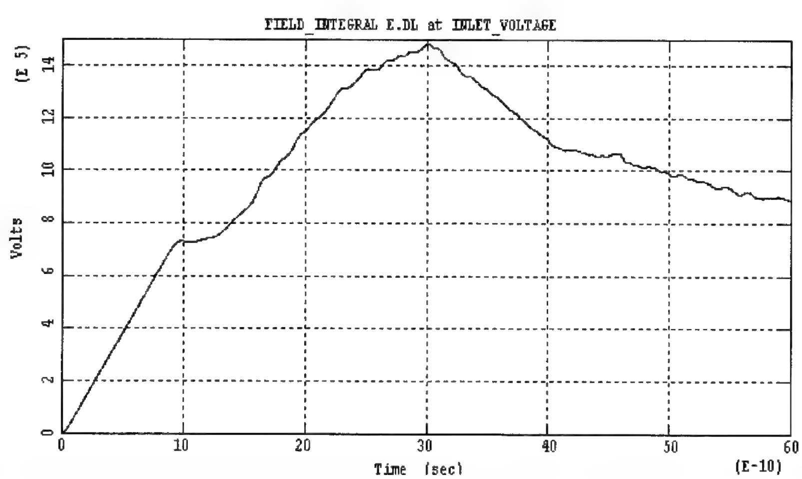
Restriction:

All objects must be of type conformal.

Examples:

To obtain the voltage over a line name, "INLET_VOLTAGE," use the following commands.

```
OBSERVE  FIELD_INTEGRAL  E.DL  INLET_VOLTAGE ;
```



This figure illustrates a time history of the voltage at the location INLET_VOLTAGE.

OBSERVE FIELD_POWER Command

Function: Specifies electromagnetic or particle power variable to be plotted vs. time.

Syntax: OBSERVE FIELD_POWER variable object
[options] ;

Arguments:

variable - power variable (see list below).
S.DA,
E.J_DRIVER.DV,
E.J_PARTICLE.DV,
E.J_OHMIC.DV,
E.J_FREESPACE.DV,
SURFACE_LOSS, (3D only)

object - name of spatial object, defined in LINE, AREA, or VOLUME command.

options - data processing options (see OBSERVE [options], Ch. 22).

Description:

The OBSERVE FIELD_POWER command is used to specify an electromagnetic and particle power variable to be plotted vs. time. The net power is nominally integrated over the specified LINE, AREA, or VOLUME. Results have the units of watts.

The power variable S.DA corresponds to the integration of the Poynting vector S over an area.

The power variable E.J_DRIVER.DV measures the instantaneous power supplied by a DRIVER command; in addition, the object name must correspond to that assigned to a particular driver.

The power variable E.J_PARTICLE.DV measures the instantaneous power gained or lost by particles from the electric field in the spatial volume specified by the object.

The power variable E.J_OHMIC.DV measures the power removed from the simulation via bulk conductivity specified by the CONDUCTANCE command; the spatial object must correspond to an object assigned the conductivity property.

The power variable E.J_FREESPACE.DV measures the power removed from the simulation through the bulk conductivity of the FREESPACE command, and thus must correspond to an object assigned the freespace property. (Note: Freespace is an artificial attenuation applied equally to the magnetic and electric fields.)

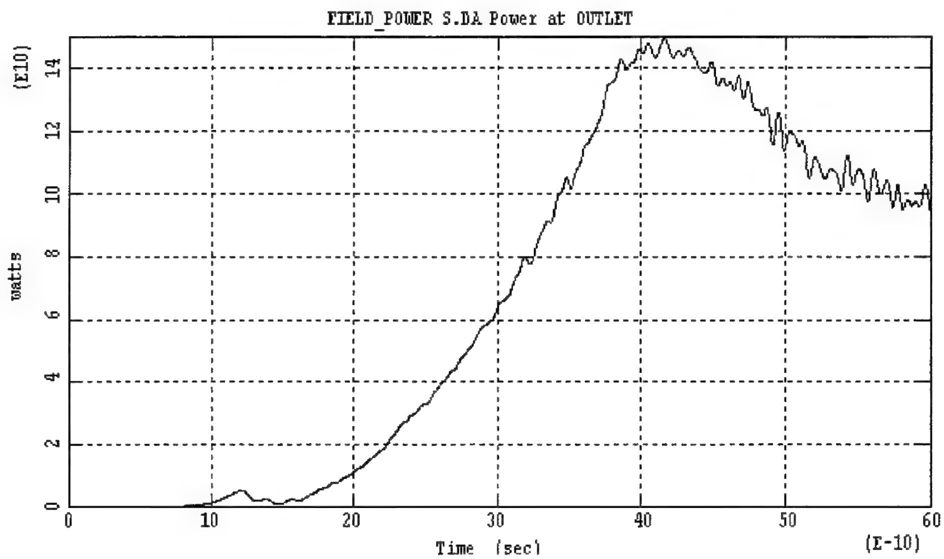
The power variable SURFACE_LOSS measures the power loss from surface currents or conductors. This power variable is only available in 3D and requires that the user invoke the SURFACE_LOSS command to activate the model.

See Also: **OBSERVE [options], Ch. 22**
CONDUCTANCE, Ch. 14
DRIVER, Ch. 19
FREESPACE, Ch. 12
SURFACE_LOSS, Ch .14

Example:

To obtain the power flow at a PORT boundary assigned the object AREA name OUTLET, you can use the following command. The figure below illustrates a typical sort of output from a 3D simulation.

```
OBSERVE FIELD_POWER S.DA OUTLET ;
```



This figure illustrates the time history of the power flow through the area OUTLET.

OBSERVE PARTICLE_STATISTICS Command

Function: Specifies a particle statistical variable to be plotted vs. time.

Syntax:

```
OBSERVE PARTICLE_STATISTICS variable species object
        [ NOTAG, TAG ]
        [options] ;
```

Arguments:

variable - particle statistics variable (see list below).
CHARGE, ENERGY,
MEAN_X, MEAN_Y, MEAN_Z,
SIGMA_X, SIGMA_Y, SIGMA_Z,
RADIUS, ELLIPTICITY,
MEAN_PX, MEAN_PY, MEAN_PZ,
SIGMA_PX, SIGMA_PY, SIGMA_PZ,
ROTATION_X, ROTATION_Y, ROTATION_Z,
MEAN_XPRIME, MEAN_YPRIME, MEAN_ZPRIME,
SIGMA_XPRIME, SIGMA_YPRIME, SIGMA_ZPRIME,
EMITTANCE_X, EMITTANCE_Y, EMITTANCE_Z,
EMITTANCE_YZ, EMITTANCE_ZX, or EMITTANCE_XY.

object - name of spatial object, defined in an AREA command in 2D or a VOLUME command in 3D.

species - name of particle species (ALL, ELECTRON, PROTON, or defined in SPECIES command).

options - data processing options (see OBSERVE [options], Ch. 22).

Description:

The OBSERVE PARTICLE_STATISTICS command is used to specify a particle statistical variable to be plotted vs. time.

The statistical quantity will be computed from all particles within the specified AREA (2D) or VOLUME (3D). This area or volume can encompass the entire collection of particles or it might just encompass a narrow slice of a continuous beam. Further restrictions can be made with the species and tag specifications. For example, in the case of a continuous beam interacting with RF of a known frequency, it might be desirable to invoke TAGGING (Ch. 24) on a single period to observe its evolution with time. To insure that all particles are used, the species should be set to ALL, and the NOTAG specification should be used. However, generally speaking, species of different charge-to-mass ratios should not be mixed.

The two variables, CHARGE and ENERGY, compute the total charge and energy within the volume, e.g.,

$$\text{CHARGE} = \sum_{macroparticles} qN \quad \text{, and}$$

$$\text{ENERGY} = \sum_{macroparticles} (\gamma-1)mc^2 N \quad ,$$

where q and m are the physical particle's charge and mass, γ is the particle's relativistic parameter, and N is the number of physical particles within each macroparticle. All other variables refer to statistical moments of sums

and products of the particles' position and momentum variables *in Cartesian coordinates*, regardless of the coordinate system used for the simulation. These statistical quantities are re-normalized to the total number of physical particles. For example, the variable MEAN_X, designated as $\langle x \rangle$, is based upon the statistical moment of the particle's X coordinate:

$$\text{MEAN\_X} = \langle x \rangle = \sum_{macroparticles} xN \Big/ \sum_{macroparticles} N \ .$$

Many of the variables find the "statistical spread" in position or momentum. For example, using the $\langle ... \rangle$ notation, the variable SIGMA_X gives the x-coordinate spread of the particles in the usual manner:

$$\text{SIGMA\_X} = \sigma_x = \langle \, (x - \langle x \rangle)^2 \rangle^{1/2} = (\langle x^2 \rangle - \langle x \rangle^2)^{1/2} \ .$$

The rotation variables are

$$\text{ROTATION\_Z} = \langle x \, p_y \rangle - \langle y \, p_x \rangle \ ,$$

and similarly for the x and y rotations. For a rigid rotating beam centered on the z-axis, this z-rotation is equal to $\Omega\langle r^2 \rangle$, where Ω is the angular rotation frequency and $\langle r^2 \rangle$ is the mean square radius.

Some variables, such as radius or emittance, assume that the collection of particles within the specified volume constitutes part or all of a beam, with the beam momentum, p_{beam}, and unit direction vector, $\hat{\mathbf{b}}$, computed from the statistical moment of the particle momenta,

$$p_{beam} \, \hat{\mathbf{b}} = \langle \mathbf{p} \rangle \ .$$

For these quantities to be computed, the actual particle distributions must be consistent with the assumption of a beam, e.g., $\langle p_x^2 \rangle + \langle p_y^2 \rangle + \langle p_z^2 \rangle >> \sigma^2_{p_x} + \sigma^2_{p_y} + \sigma^2_{p_z}$. If this criterion is not met, then a value of zero will be displayed for the beam's statistical variable. The beam direction, $\hat{\mathbf{b}}$, is needed to compute the RMS radius, given in tensor form by

$$\text{RADIUS} = \langle r^2 \rangle^{1/2} = \langle \, (\mathbf{x} - \langle \mathbf{x} \rangle) \cdot [\underline{1} - \hat{\mathbf{b}} \, \hat{\mathbf{b}} \,] \cdot (\mathbf{x} - \langle \mathbf{x} \rangle) \, \rangle^{1/2} \ .$$

Ellipticity indicates how much a beam diverges from a round shape. For a beam centered on the z-axis, ellipticity has units of length, and is defined here as

$$\text{ELLIPTICITY} = 3^{1/2} \ (\ (\langle x^2 \rangle - \langle y^2 \rangle)^2 + 4\langle xy \rangle^2 \ )^{1/4} \ .$$

This quantity is invariant to rotation in the x-y plane. The diagnostic uses the equivalent tensor invariant, generalized for arbitrary beam axis.

The beam direction, $\hat{\mathbf{b}}$, is also employed in the "primed-momenta" used for emittance calculation, e.g., a particle's x-primed momentum is defined as:

$$x' = p_x / \, \hat{\mathbf{b}} \cdot \mathbf{p} \ ,$$

and similarly for y' and z'. Physically, the primed momenta represents the tangent of the angle that a particle trajectory makes with the beam axis. The standard RMS emittance formula is $(\varepsilon_{xx'}/4)^2 = \langle x^2 \rangle \langle x'^2 \rangle - \langle xx' \rangle^2$ and is

based upon an assumption of a beam centered on the z-axis, e.g., satisfying $\langle x\rangle=0$ and $\langle x'\rangle=0$. For arbitrary axis location and direction, where $\langle x\rangle\neq0$ and/or $\langle x'\rangle\neq0$, the RMS emittance formula is generalized here to be:

$$\text{EMITTANCE\_X} = \varepsilon_{xx} = 4\left(\langle(x-\langle x\rangle)^2\rangle\langle(x'-\langle x'\rangle)^2\rangle - \langle(x-\langle x\rangle)(x'-\langle x'\rangle)\rangle^2\right)^{1/2},$$

and similarly for ε_{yy} and ε_{zz}. Note that for a beam direction along the z-axis, the above definitions result in $\varepsilon_{zz}=0$, exactly. Hence, this definition provides only the transverse emittance. For oblique beam axes the emittance is a symmetric tensor, and additional cross-term emittances become important,

$$\text{EMITTANCE\_XY} = \varepsilon_{xy} = \varepsilon_{yx} = 4\left(\langle(x-\langle x\rangle)(y-\langle y\rangle)\rangle\langle(x'-\langle x'\rangle)(y'-\langle y'\rangle)\rangle - \langle(x-\langle x\rangle)(y'-\langle y'\rangle)\rangle\langle(x'-\langle x'\rangle)(y-\langle y\rangle)\rangle\right)^{1/2},$$

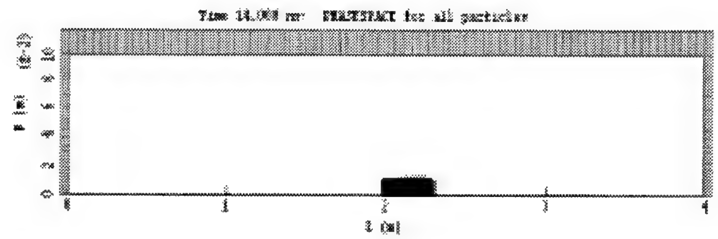
and similarly for $\varepsilon_{yz'}$, and $\varepsilon_{zx'}$. Care must be taken when interpreting emittance of a rotating beam. Notice that for a beam centered on the z-axis, the cross-term emittance takes on an approximate value $\varepsilon_{xy}=2\langle r^2\rangle\Omega/p_{beam}$, where Ω is the angular rotation frequency. However, if the rotation is due to immersion in a magnetic field, then it is not true emittance, because it is possible to remove it by including the vector potential in the momentum. This diagnostic will report the false rotation emittance, since no vector potential correction is made.

See Also: **OBSERVE [options], Ch. 22**

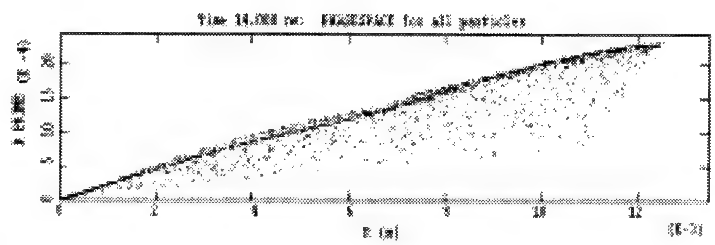
Example:

An electron bunch is emitted at full energy from the left wall of a long drift region. It is desired to determine the space-charge-induced growth in radius and emittance of the bunch. A 3D simulation in cylindrical coordinates is performed, and the output is shown below. At 14 nanoseconds, the bunch is midway through the drift region, as indicated in the final PHASESPACE plot. An R-R′ PHASESPACE plot, in the upper right, shows the traditional emittance phase space and is produced with the help of the user-defined functional phase-space variable, R_Prime. An eyeball estimate from this plot of the total area occupied by the particles suggests an actual emittance value around 8.5 micro-radian-meters. The initial bunch has uniform density, out to a radius of 1 cm; hence, the initial RMS radius is 0.707 cm. After 20 nanoseconds, the RMS radius has grown to 1.05 cm. The initial RMS x-emittance of the fully formed bunch is about 1.0 micro-radian-meters. After 20 nanoseconds, the RMS x-emittance has grown to about 4.6 micro-radian-meters. At 14 nanoseconds, the RMS x-emittance is about 3.7 micro-radian-meters, which can be compared to our eyeball estimate of 8.5 micro-radian-meters actual emittance, the difference being accounted for by a factor of 0.707 going from R-R′ to X-X′ phase space, and additional reduction to go from actual to RMS value. The following commands were used to produce the four plots.

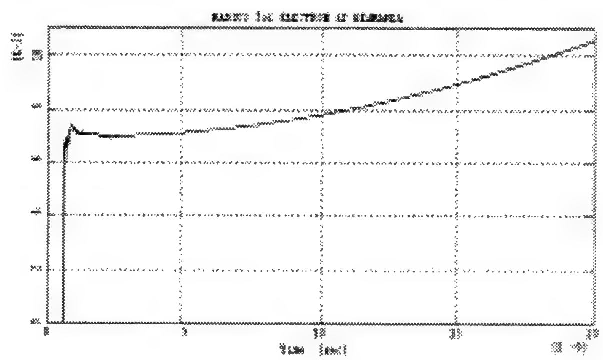
```
FUNCTION R_Prime(Z,R,THETA,PZ,PR,PTHETA) = PR/PZ ;
PHASESPACE AXES X1    X2      timerPhase;
PHASESPACE AXES X2 R_Prime timerPhase;

OBSERVE PARTICLE_STATISTICS RADIUS         ELECTRON beamArea NOTAG;
OBSERVE PARTICLE_STATISTICS EMITTANCE_X    ELECTRON beamArea NOTAG;
```



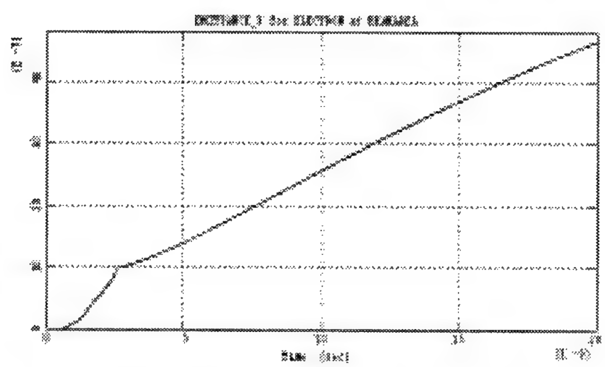
This PHASESPACE figure shows a single electron bunch midway through the drift space.



This figure shows the traditional r′-r PHASESPACE associated with emittance.



This figure shows the expansion of the electron beam radius as a function of time.



This figure shows the growth of the emittance of an electron bunch as it travels through the drift region.

OBSERVE RESONANT_PORT Command

Function: Specifies resonant_port variable to be plotted vs. time.

Syntax: OBSERVE RESONANT_PORT resonant_port variable [options] ;

Arguments:

resonant_port - resonant_port line (2-D) or area (3-D), used in RESONANT_PORT command.

variable - resonant_port variable, e.g.,

V_LINE, V_LINE_REAL, V_LINE_IMAG, V_LINE_PHASE,
V_GAP, V_GAP_REAL, V_GAP_IMAG, V_GAP_PHASE,
I_BEAM, I_BEAM_REAL, I_BEAM_IMAG, I_BEAM_PHASE,
I_GAP, I_GAP_REAL, I_GAP_IMAG, I_GAP_PHASE,
N*I_IN, N*I_IN_REAL, N*I_IN_IMAG, N*I_IN_PHASE,
BEAM_POWER, BEAM_POWER_CYCLING,
GAP_POWER, GAP_POWER_CYCLING,
INPUT_POWER, INPUT_POWER_CYCLING,
R_CAVITY_POWER,
INDUCTIVE_POWER_CYCLING,
CAPACITIVE_POWER_CYCLING,
R_EXTERNAL_POWER,

options - processing options (see OBSERVE [options]).

Description:

The command, OBSERVE RESONANT_PORT, is used to specify a resonant_port variable to be plotted vs. time. The resonant_port line or area is the same as what is used in a RESONANT_PORT command. The variable specifies the resonant_port variable to be observed.

Many of resonant_port's internal quantities are actually complex numbers, having both real and imaginary parts, or alternatively magnitude and phase. These complex quantities are observed by one of four different variables, giving magnitude, real-part, imaginary-part, or complex phase. Included in this class of variables are, V_LINE, the voltage across the voltage_line, V_GAP, the voltage across the resonant_port boundary, I_BEAM, the beam current, I_GAP, the gap current used in the alternative model, and N*I_IN, the ideal external current:

V_LINE,	V_LINE_REAL,	V_LINE_IMAG,	V_LINE_PHASE,
V_GAP,	V_GAP_REAL,	V_GAP_IMAG,	V_GAP_PHASE,
I_BEAM,	I_BEAM_REAL,	I_BEAM_IMAG,	I_BEAM_PHASE,
I_GAP,	I_GAP_REAL,	I_GAP_IMAG,	I_GAP_PHASE,
N*I_IN,	N*I_IN_REAL,	N*I_IN_IMAG,	N*I_IN_PHASE.

A second class of variables provides the power and energy within the circuit modeled by a resonant port. These quantities are also complex; however, the real power indicates actual power, while the imaginary part indicates $\omega * U$, where ω is the 2π times the frequency and U represents the energy which is cycled into and out of

the energy source within one period. This class of variables includes the BEAM_POWER = ½ $I_{beam}^* V_{gap}$, the GAP_POWER = ½ $I_{gap}^* V_{gap}$, and the INPUT_POWER = ½ $nI_{in}^* V_{gap}$:

BEAM_POWER,	BEAM_POWER_CYCLING,
GAP_POWER,	GAP_POWER_CYCLING,
INPUT_POWER,	INPUT_POWER_CYCLING.

The last class of variables represents additional power and energy variables that are either pure real or pure imaginary. These are:

R_CAVITY_POWER	$= ½ \lvert V_{gap}\rvert^2 / R$,
INDUCTIVE_POWER_CYCLING	$= ½ \lvert V_{gap}\rvert^2 / (i\omega L)$,
CAPACITIVE_POWER_CYCLING	$= ½ \lvert V_{gap}\rvert^2 (i\omega C)$,
R_EXTERNAL_POWER	$= ½ \lvert V_{gap}\rvert^2 / R_{external}$.

See Also: **RESONANT_PORT, Ch. 12**
OBSERVE [options], Ch. 22

OBSERVE STRUT Command

Function: Specifies strut variable to be plotted vs. time.

Syntax: OBSERVE STRUT object variable { point, line } [options] ;

Arguments:

object	-	object name, defined in STRUT command.
variable	-	strut variable (See STRUT command), CURRENT, CHARGE, INDUCTANCE, RESISTANCE, STORED_ENERGY, or OHMIC LOSS.
point	-	name of point, defined in POINT command.
line	-	name of line, defined in LINE command.
options	-	processing options (see OBSERVE [options]).

Description:

The command, OBSERVE STRUT, is used to specify a strut variable to be plotted vs. time. The strut name is defined in a STRUT command. The variable specifies the strut variable to be observed. The point or line must lie along the specified strut. If the spatial object is a point, the variable at that point will be recorded. If the spatial object is a line, an average or integral along the line will be taken, whichever makes physical sense.

See Also: **STRUT, Ch. 15**
OBSERVE [options], Ch. 22

OBSERVE TRAMLINE Command

Function: Specifies a transmission-line variable to be plotted vs. time in a 2D simulation.

Syntax: OBSERVE TRAMLINE tline_name variable point
[options] ;

Arguments:

tline_name	-	transmission-line name, defined in TRAMLINE command.
variable	-	transmission-line variable (see TRAMLINE, Ch. 13), VOLTAGE, CURRENT, POWER, CAPACITANCE, INDUCTANCE, IMPEDANCE, ENERGY-CAP, ENERGY-IND, or ENERGY-TOT.
point	-	name of spatial point on transmission line coordinate, defined in POINT command.
options	-	processing options (see OBSERVE [options], Ch. 22).

Description:

The OBSERVE TRAMLINE command is used to specify a transmission-line variable to be plotted vs. time. The tline_name specifies the transmission line and the point specifies the coordinate location. The variable specifies the transmission line variable to be observed.

See Also: TRAMLINE, Ch. 13
OBSERVE [options], Ch. 22

Restrictions:

This command is available only in 2D simulations.

This page is intentionally left blank.

23. 1D PLOTS

This Chapter covers the following commands:

(2D and 3D simulations)	(2D simulations)
RANGE [options]	RANGE TRAMLINE
RANGE FIELD	
RANGE FIELD_INTEGRAL	
RANGE FIELD_POWER	
RANGE HISTOGRAM	
RANGE PARTICLE	

You can use the RANGE commands to plot the value of a simulation variable vs. a spatial dimension. Typically, the spatial dimension will be one of the spatial coordinates (x1, x2, or x3) or perhaps a limited portion of a coordinate. In other cases, it may represent an arbitrary path in space (such as the inner surface of a particle collector).

There are many data-processing operations that can be applied to RANGE data, and these are described in the RANGE [options] command. Defaults are set for many of these options. All of the other RANGE commands involve plotting a particular measurement_type, such as FIELD, etc. Some measurement_types may apply only to 2D or 3D simulations.

RANGE [options] Command

Function: Specifies data-processing options for RANGE commands.

Syntax:

```
RANGE measurement_type variable [ arguments ] timer
      [ MOVIE ]
      [ AXIS { X, Y } minimum maximum [ step ] ]
      [ PRETRANSFORM function(x,y,t) ]
      [ TRANSFORM function(x,y) ]
      [ FFT { MAGNITUDE, COMPLEX }]
      [ { PLOT, NOPLOT } ]
      [ { NOPRINT, PRINT } ]
      [ { NODUMP, DUMP } ] ;
```

Arguments:

measurement_type - type of mesurement variable:
The following measurement types FIELD, FIELD_POWER, PARTICLE, and HISTOGRAM are available in 2D and 3D simulations. The measurement type TRAMLINE is only available in 2D simulations.

variable - differs for each variable_type, see other RANGE commands.
arguments - differs for each variable_type, see other RANGE commands.
timer - timer name, defined in TIMER command.
minimum, ... - plot boundaries for position axis (X) or variable axis (Y).
step - plot subdivisions for position axis (X) or variable axis (Y).
function - transformation function, defined in FUNCTION command.
x - dummy argument representing position.
y - dummy argument representing the variable.
t - dummy argument representing simulation time.

Defaults:

The default values for the data processing options are listed in the table below. Default operations are automatically applied to RANGE commands, even if the specific operation is not explicitly entered in the command. For example, if you reset the FFT index default from 1 to 2, all subsequent RANGE commands will automatically trigger an FFT operation, even if FFT is not entered in those commands.

Keyword	Argument	Default
AXIS X	Minimum, maximum, step	(entire simulation coordinate)
AXIS Y	Minimum, maximum, step	(determined by variable range)
PRETRANSFORM	Function	unity (no transformation)
TRANSFORM	Function	unity (no transformation)
{ PLOT, NOPLOT }	(none)	PLOT (output is plotted)
{ NODUMP, DUMP }	(none)	NODUMP (output is not dumped)
{ NOPRINT, PRINT }	(none)	NOPRINT (output is not printed)

Description:

The RANGE command provides the capability to output a 1D plot, i.e., a plot of a simulation variable vs. spatial position at times specified by a timer. The spatial position will usually be measured along one of the simulation axes but may, in certain circumstances, contain multiple segments and bends. (For example, to plot energy deposition on a curved surface, the spatial distance would be measured along the perimeter of the surface.) Separate RANGE commands are structured for each variable_type (e.g., RANGE FIELD for field variables, etc.), since each variable_type may have unique arguments. Some variable_types may apply only to 2D or 3D simulations. The RANGE commands describing each variable_type and its unique arguments follow later in this Chapter. The present RANGE [options] command describes the general data-processing operations (e.g., FFT) which are common to all RANGE commands.

The MOVIE option permits you to generate a sequence of bitmaps (in PCX format) from a sequence of RANGE plots. In order for this sequence to provide an appealing visual representation of the data, you should also use the AXIS option to enforce limits. When the MOVIE option is invoked, MAGIC generates a series of bitmaps that are saved in a folder. There is one movie folder per command that uses the MOVIE option. You can "play" the movie by using the MOVIE command. Alternatively, you can post process the files to create an AVI file.

The AXIS option allows refinement of the actual plot boundaries, both in extent (minimum and maximum) and in the size of plot subdivisions (step). The arguments, AXIS X, refer to the independent spatial dimension, which is always plotted horizontally as the x-axis in meters. The arguments, AXIS Y, refer to the dependent simulation variable, which is always plotted vertically as the y-axis. The units of the variable axis depend upon the variable as well as any data-processing options selected.

There are three, optional, data-processing operations: PRETRANSFORM, TRANSFORM, and FFT. (When you specify more than one operation, please be aware that the operations will be performed in this order, and not in the order you place them in the command.) Each operation causes the variable data to be modified in a particular fashion.

As an adjunct to these three operations, you can also use the INTEGRATE option in a TIMER command to perform temporal averaging or integration over an interval prior to the time at which output occurs. (A common use of this option is to average particle or Poynting data over a full RF period to get cycle-averaged quantities.) When this option is used, the integration will be performed after the PRETRANSFORM operation and before the TRANSFORM operation (if either is specified).

The PRETRANSFORM option causes a transformation of the variable <u>before</u> it is temporally averaged. (Normally, this option would be used only if you intend to perform temporal averaging with the TIMER command.) The function has three dummy arguments: x. y, and t. (A common use of pretransformation is to multiply the variable by a sinusoidal time function to extract harmonic information.) The name of the pretransformation function will appear on the y-axis of the plot.

The TRANSFORM option causes a transformation of the variable <u>after</u> temporal averaging (if any). Thus, this function has only two arguments: x and variable. The name of the pretransformation function will appear on the y-axis of the plot instead of the name from the pretransformation function (if any).

The FFT option is used to specify a Fourier transform. The default is NOFFT. You can specify a plot of either the magnitude or the complex parts of the FFT. The default is MAGNITUDE. There is no ability to control the boundaries of the FFT plots.

The actual FFT calculation proceeds as follows:

$$\tilde{y}(\lambda) = 2\int_{x_{\min}}^{x_{\max}} dx\, y(x)\, e^{-i2\pi\lambda x} \quad .$$

For reference, the inverse FFT is:

$$y(x) = \int_0^{\lambda_{\max}} d\lambda\, \mathrm{Re}\left\{\tilde{y}(\lambda)\, e^{i2\pi\lambda x}\right\} \quad .$$

The conventional factor of 2π is missing because of the use of inverse wavelength, λ, instead of wave number. The extra factor of two in the transform is compensated for in the inverse by integrating over only positive inverse wavelengths, a common manipulation where pure real data is concerned.

The PLOT, DUMP, and PRINT options allow 1D plot results to be displayed and/or stored by three different methods:

1) as a graphical plot in a metafile or on screen,
2) as a data record in the GRD file, or
3) as printed column data in the LOG file.

A 1D plot on metafile or on screen is the normal result of the RANGE command. Under certain circumstances, it may be desirable to have particular plots displayed, but not stored to save disk space, or perhaps stored, but not displayed, for some other reason. The PLOT, NOPLOT, DUMP, and NODUMP options can be applied to individual 1D plots; i.e., it is not necessary to treat all the plots in the same way. Alternatively, DUMP TYPE RANGE or DUMP TYPE ALL commands can be used to record all 1D plots in the GRD file (DUMP, Ch. 25).

The PRINT option can be used to generate tables of 1D plot data in the LOG file. This option cannot be applied separately to individual 1D plots. Therefore, the last PRINT or NOPRINT option entered will govern printing for all RANGE commands to the LOG file.

Restrictions:

The total number of RANGE commands defined in a simulation is limited to fifty.

See Also: TIMER, Ch. 11
RANGE FIELD, Ch. 23
RANGE FIELD_ENERGY, Ch. 23
RANGE HISTOGRAM, Ch. 23
RANGE PARTICLE, Ch. 23
RANGE TRAMLINE, Ch. 23

RANGE FIELD Command

Function: Plots field variable vs. spatial position.

Syntax: RANGE FIELD field line timer [options] ;

Arguments:

field - field variable (see list below).
The following field options are available in both 2D and 3D simulations: B1, B2, B3, E1, E2, E3, J1, J2, J3,Q0, B1ST, B2ST, B3ST, B1AV, B2AV, B3AV, E1AV, E2AV, and E3AV. The following field options area available only in 2D simulations: D1, D2, D3, DIE1, DIE2, DIE3, SIGMA, QERR, E1ST, E2ST, E3ST, and PHST.

line - name of conformal line, defined in LINE command.

timer - name of timer, defined in TIMER command.

options - data-processing options (RANGE [options], Ch. 23).

Description:

This command produces a 1D plot of a field variable along a specified spatial line. The field variable may be any of those specified above. Note that some fields are available only in 2D simulations. The spatial line must be conformal.

Restrictions:

1. Some fields are defined only in 2D simulations and are not available in 3D.
2. The spatial line must be conformal.
3. The total number of RANGE commands is limited to fifty.

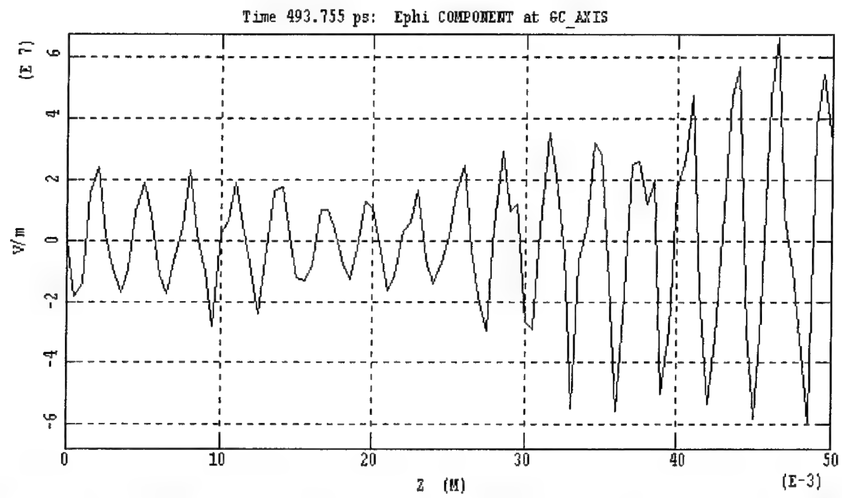
See Also:

LINE, Ch. 10
TIMER, Ch. 11
RANGE [options], Ch. 23

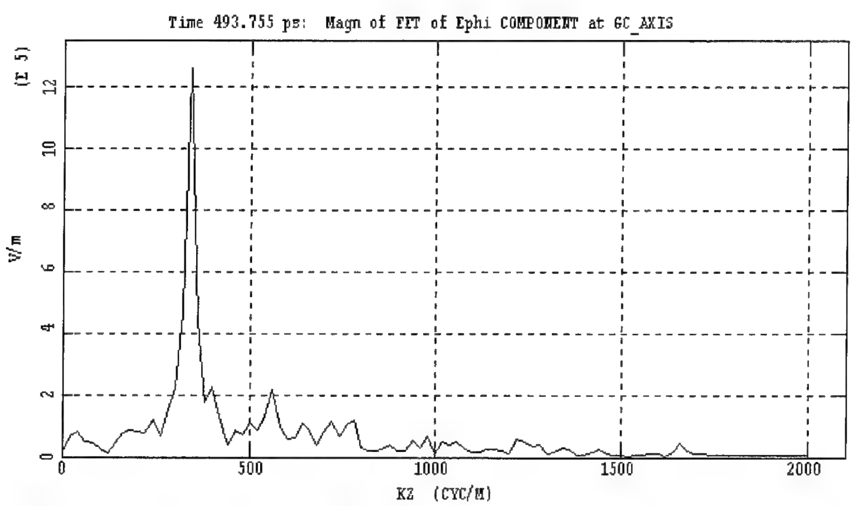
Examples:

The following commands illustrate the use of the RANGE options in diagnosing a cylindrical CARM simulation. A gyro-magnetic beam with a current of 2.5 kA is introduced at the left end of a hollow, confining chamber. A confining magnetic field of 2.024 tesla is applied along the axial direction. A current-density driver with a frequency of 103.29 GHz is applied at the emission surface. The RANGE command is used to measure the magnitude of the azimuthal electric field at a fixed value of r versus z and performs an FFT on the data. The following figures show the results at a simulation time of 0.29 ns.

```
TIMER end_of_run DISCRETE REAL .494nanosec ;
LINE GC_axis CONFORMAL  0.0, Radius_Center SYS$X1MX, Radius_Center ;
RANGE FIELD E3 GC_axis end_of_run  FFT MAGNITUDE ;
```



Range plot of Ephi electric field showing the periodic variation in field intensity and the amplification of the signal with increasing axial coordinate.



Spatial FFT of the Ephi electric field. Note that only one strong spatial component is evident in the spectrum.

RANGE FIELD_INTEGRAL Command

Function: Specifies electromagnetic field integral to be plotted vs. position.

Syntax: RANGE FIELD_INTEGRAL variable {X1, X2, X3} {area,volume} timer [options] ;

Arguments:

variable	- field integral variable, currently only E.DL (volts) and J.DA (amps) are available.
area,volume	- conformal area/volume enclosing the measurement region, defined in the AREA command for 2D simulations and E.DL in 3D simulations or defined in the VOLUME command for J.DA in 3D simulations.
timer	- name of timer, defined in TIMER command.
options	- data-processing options (RANGE [options], Ch. 23).

Description:

The RANGE FIELD_INTEGRAL command plots an electromagnetic field integral variable versus spatial position. Currently, only voltage and current are treated. The user must specify the direction, X1, X2, or X3, over which to plot the field integral variable. The area/volume specifies the spatial region of interest. For the E.DL variable, the direction of integration is the direction in the plane of the specified area that is transverse to the plotting direction. For the J.DA variable, the normal to the area of integration is in the plotting direction. The integral will carry the combined units of the field and the spatial differential, DL (m), DA (m^2). The E.DL has units of volts and J.DA has units of amps. The value given for E.DL has the correct sign for a voltage measurement; i.e, it is actually the negative of the E.DL integral.

Restrictions:

1. The total number of RANGE commands defined in a simulation is limited to fifty.

See Also: **TIMER, Ch. 11**
RANGE [options], Ch. 23

Examples:

To measure the axial current for a cylindrical simulation within the spatial object "coaxialCavity" intervals of 100 time steps, use the following commands,

```
TIMER currentTimer PERIODIC 100 9999 100;
RANGE FIELD_INTEGRAL J.DA X1 coaxialCavity currentTimer;
```

RANGE FIELD_POWER Command

Function: Specifies electromagnetic power variable to be plotted vs. position.

Syntax: RANGE FIELD_POWER variable {X1, X2, X3} {area,volume} timer
[FREQUENCY n_f f_1 f_2 ... f_{nf}]
[PHASE_VELOCITY { velocity(x,f), SECTIONS n_s x_1 x_2 ... x_{ns} [SMOOTH] }]
[options] ;

Arguments:

variable	- power variable, currently only S.DA (watts) is available.
area,volume	- conformal area/volume enclosing the measurement region, defined in the AREA command for 2D simulations or defined in the VOLUME command for 3D simulations.
timer	- name of timer, defined in TIMER command.
n_f	- number of frequencies in list (integer).
f_1 f_2 ... f_{nf}	- list of frequencies (Hz).
velocity	- phase velocity (m/sec), constant or defined in FUNCTION command.
n_s	- number of section breaks for computed phase velocity (integer).
x_1 x_2 ... x_{ns}	- section breaks for computed phase velocity (unitless).
options	- data-processing options (RANGE [options], Ch. 23).

Description:

The RANGE FIELD_POWER command plots an electromagnetic power variable vs. spatial position. Currently, only the Poynting flux power variable is treated. The user must specify the direction, X1, X2, or X3, over which to plot the power variable. The area/volume specifies the spatial region of interest, with the power variable being integrated over this area/volume in the direction transverse to the plotting direction.

It is possible to divide the power variable into its constituent frequency components, using the FREQUENCY option (2D only). This additional analysis is based upon Parseval's identity, and uses the Fourier transforms of the individual electromagnetic fields comprising the power variable. To perform its function properly, the FREQUENCY option requires data over several (at least two) oscillation periods of the smallest frequency of interest. This means that the TIMER ... INTEGRATE option <u>must be used, with an integration interval of several such periods</u>. The user must supply the frequencies of interest by listing the frequencies in order from smallest to largest. A value of zero for f_1 is allowed. For purposes of determining the length of the timer integration interval, the smallest frequency of interest is the smaller of either the lowest non-zero frequency or the minimum spacing between frequencies, if more than one frequency is requested. When the FREQUENCY option is used, more than one plot is produced, the first being the total power, and the subsequent plots being the power in each frequency, one plot per specified frequency.

The FREQUENCY analysis requires that the entire time history of the fields be stored to a file on disk before the analysis can be performed, resulting in a sizable scratch file. For this reason, the maximum amount of information in terms of frequency content should be derived from the fewest number of RANGE FIELD_POWER ... FREQUENCY commands possible.

For the Poynting flux power variable only, an additional division into positive power flow and negative power flow directions is done whenever timer integration is performed (2D only). This directional analysis is performed on each non-zero frequency component individually if the FREQUENCY option is also specified. The Poynting power flux before directional division may be of either sign. The directional division is pure, that is, positive power is always positive in sign and negative power is always negative in sign; however, RANGE reverses the sign of the negative power on the plot to facilitate reading of the data. The result of the directional analysis is three additional plots for each non-zero frequency, one showing positive power, another showing negative power, and a third showing the phase velocity used. For example, if five non-zero frequencies are requested, then 16 (=1+4x5) range plots will result.

The directional analysis requires that an estimate of the phase velocity be made. The default internal algorithm used to estimate the phase velocity is based upon a minimum standing-wave principle which is of unknown robustness. Hence, if the phase velocity of the system can be established a priori, then it is advisable to use the PHASE_VELOCITY option to supply this quantity directly. Note that the phase velocity is, in general, a function of frequency and position along the data range. Even when the phase velocity is not known a priori, the user must assist the internal algorithm in cases where the phase velocity is known to vary spatially by breaking up the spatial range into sections. The SECTIONS option requires the user to specify n_s positions, $x_1 \ldots x_{ns}$, where section breaks occur. By default, the phase velocity is assumed to be constant between the section breaks. If, instead, a smooth tapering in phase velocity is more appropriate, then the SMOOTH option can be invoked. When the PHASE_VELOCITY option is not employed, the entire spatial range is assumed to have a constant phase velocity by default.

Restrictions:

1. The total number of RANGE commands defined in a simulation is limited to fifty.

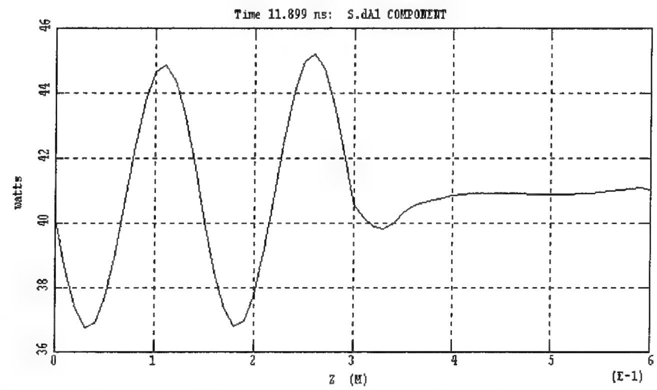
2. The FREQUENCY analysis currently works only for the S.DA variable in 2D simulations.

See Also: **TIMER, Ch. 11**
RANGE [options], Ch. 23

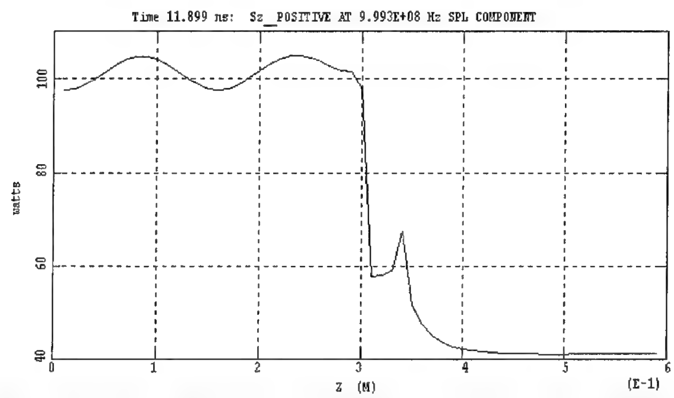
Examples:

In this example, a RANGE FIELD_POWER command is used to display the power flow in a coaxial transmission line which has an obstruction halfway down its length. A 100-watt, 1 GHz signal is introduced at the left port of the coaxial line. At the obstruction, approximately 59% of the power is reflected, while the remaining 41% is transmitted and continues to propagate towards the PORT boundary on the right. The net power, Figure 23-2a, throughout the transmission line is therefore 41 watts; however, the directional analysis, Figures 23-2b and c, properly shows that, to the left of the obstruction, the power consists of 100-watt forward power and 59-watt backward power. It also shows that the power to the right of the obstruction is all forward power. The TIMER command is also shown below, because the directional analysis requires that the timer INTEGRATE option be used. Here the time integration is performed over a single oscillation period.

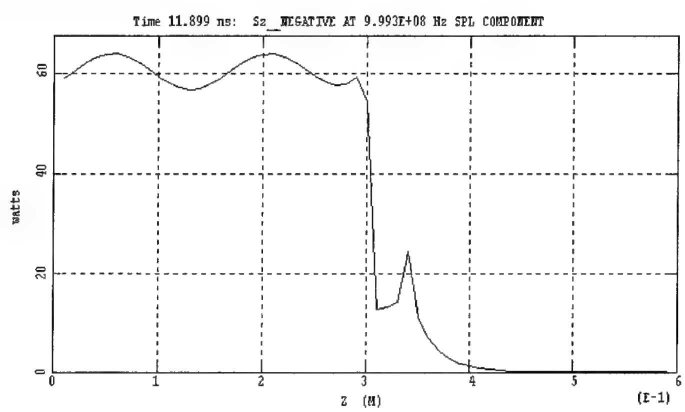
```
RFP2=2.*RF.PERIOD ;
     TIMER CHECK.PWR PERIODIC REAL TSTART SYS$RUNTIME RFP2 INTEGRATE RFP2 ;
     FLO  = RF.FREQUENCY ;
     FHI  = 2*RF.FREQUENCY ;
     RANGE FIELD_POWER S.DA X1 INTERIOR CHECK.PWR FREQUENCY 2 0 FLO ;
```



Range plot of the RF-cycle averaged net power in a coaxial cable with an obstacle in the line that causes signal reflections. Note that the mean net power is about 41 watts.



Range plot of the RF-cycle averaged forward power in a coaxial cable with an obstacle in the line that causes signal reflections. Notice that the mean supplied power at the left side is 100 watts, and the mean power that makes it to the end of the coaxial line is 41 watts.



Range plot of the RF-cycle averaged backward power in a coaxial cable with an obstacle in the line that causes signal reflections. Note that the mean return power is about 59 watts.

RANGE PARTICLE Command

Function: Plots particle variable vs. spatial position.

Syntax:

```
RANGE  PARTICLE variable  species  { X1, X2, X3 }  timer
        [ WEIGHTING  {  flux, density  }  ]
        [  options  ]
```

Arguments:

variable	- particle variable (see list below). CURRENT – net current (A) POWER – net power (watts). ENERGY – average energy per particle (eV).
species	- particle species name, ELECTRON< PROTON< or defined in SPECIES command.
timer	- name of timer, defined in TIMER command.
function	- function to convert energy to efficiency, defined in FUNCTION command.
x	- dummy argument representing position.
ke	- dummy argument representing kinetic energy.
options	- data –processing options (RANGE [options], Ch. 23).

Description:

This command produces a 1D plot of a particle variable along a specified coordinate axis. The particle variable may be current, power, or energy.

The EFFICIENCY option requires a function which will be used to convert energy/particle to efficiency. The function has two arguments: x (the axis coordinate) and ke (the value of the energy at x). For example, if the beam has 60 keV of energy prior to the output circuit and the collector, the conversion function could be defined as f(x,ke) = 1 – w/60.

The WEIGHTING option can be used to compute energy per particle by two methods. FLUX weighting (the default) derives energy per particle as a ration of energy flux to power flux, whereas DENSITY weighting derives energy per particle as a ration of energy density to charge density. The two methods may produce subtle differences when the beam is not steady-state or CW, when it contains counter-flowing particles, or when particle losses are occurring.

Restriction:

The total number of RANGE commands is limited to fifty.

See Also:

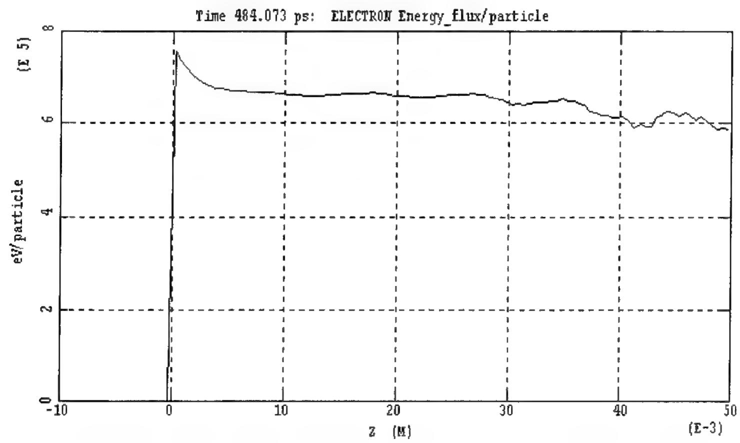
LINE, Ch. 10
TIMER, Ch. 11
RANGE [options], Ch. 23

Examples:

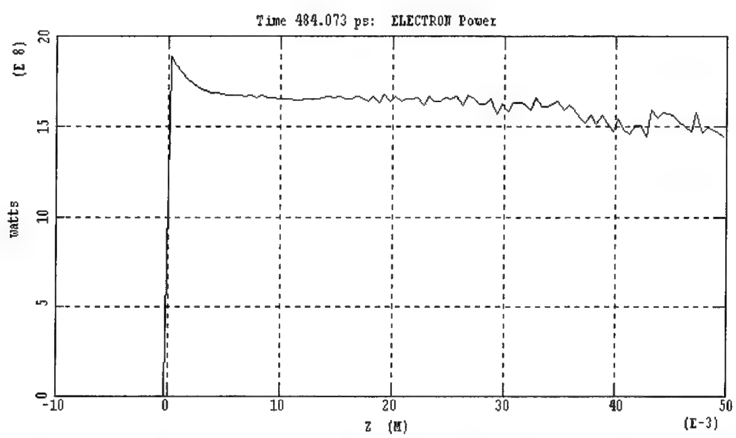
The following commands illustrate the use of the RANGE options in diagnosing a cylindrical CARM simulation. A gyro-magnetic field of 2.204 tesla is applied along the axial direction. A current-density driver with a frequency of 103.29 GHz is applied at the emission surface. The following RANGE commands measure

the transverse averaged particle energy and power versus z. A temporal average over one period of the drive frequency is applied by using a TMER with an integration interval equal to the period. The following figures show the plots resulting from these commands for a simulation time of 0.29 ns.

```
TIMER TAVERAGED PERIODIC REAL PERIOD10 1 PERIOD10 INTEGRATE PERIOD;
RANGE PARTICLE POWER ELECTRON X1 TAVERAGED;
RANGE PARTICLE ENERGY ELECTRON X1 TAVERAGED;
```



Range plot of time-averaged particle energy in eV versus axial coordinate.



Range plot of time-averaged electron beam power as a function of axial position. Note the loss of power in the electron with increasing z. This power is delivered up to the electric field.

RANGE HISTOGRAM Command

Function: Specifies histogram of particle distribution versus one phase space variable.

Syntax: RANGE HISTOGRAM species coordinate nbins bin_lo bin_hi timer
[WINDOW coordinate minimum maximum]
[options] ;

Arguments:

species	- particle species (ELECTRON, PROTON, or defined in SPECIES command).
coordinate	- particle phase space variable (X1, X2, X3, P1, P2, P3, KE, or VELOCITY.
nbins	- number of bins for histogram
bin_lo	- lower end of histogram.
bin_hi	- lower end of histogram.
minimum	- lower end of sampling window.
maximum	- upper end of sampling window.
timer	- timer name, defined in TIMER command
options	- data-processing options (RANGE [options], Ch. 23).

Description:

The RANGE HISTOGRAM command produces a histogram of the particle distribution function versus one of the particle phase space coordinates at times specified by the timer. The timer must specify times which take into account the kinematics step_multiple (LORENTZ, Ch. 18).

The argument, coordinate, specifies a phase-space variable. The standard variables that may be specified are X1, X2, X3, P1, P2, or P3. These are the physical coordinates and momenta of the particles. In addition, the arguments, KE, and VELOCITY may be used to select the kinetic energy per physical particle (eV) and the magnitude of the velocity (m/s).

The keyword, WINDOW, can be used to select the acceptance window in any of these variables: x_1, x_2, x_3, p_1, p_2, p_3, ke, and v. Only particles falling within the variable limits, minimum and maximum, are included in the histogram sample space. Note that the window coordinates allow you to specify the sample space of your simulation.

Restriction:

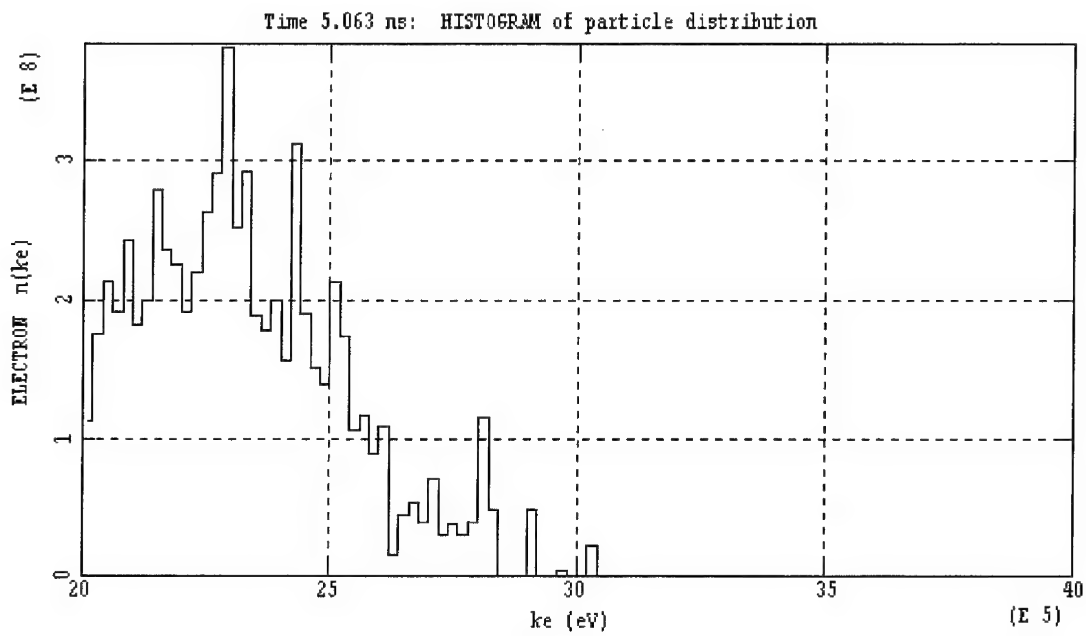
The total number of RANGE commands defined in a simulation is limited to fifty.

See Also: TIMER, Ch. 11
RANGE [options], Ch. 23

Examples:

The following commands illustrate the use of the RANGE HISTOGRAM command to sample a particle distribution in a simulation of a 3-MVolt diode. The WINDOW option is used to restrict the sample to the region of the device that is physically located downstream from the accelerating electrode. The first RANGE command examines the velocity spread of the beam electrons. The plot will include the average velocity of the sampled electrons as well as the RMS value. The second RANGE command is used to examine the kinetic energy of the sampled electrons. The third RANGE command is used to sample the axial momentum component of the electrons, and the final RANGE command is used to examine the radial profile of the sample electrons. In each case, the average value and the RMS value of the sampled variable is evaluated and printed as part of the plot informative text.

```
RANGE HISTOGRAM ELECTRON VELOCITY 100 2.9e8 3E8  CHKIT
          WINDOW X1 ZMAX_ANODEELECTRODE ZMAX_ENDPLATE ;
RANGE HISTOGRAM ELECTRON KE 100 2E6 4E6  CHKIT
          WINDOW X1 ZMAX_ANODEELECTRODE ZMAX_ENDPLATE ;
RANGE HISTOGRAM ELECTRON P1 100 1E9 3E9  CHKIT
          WINDOW X1 ZMAX_ANODEELECTRODE ZMAX_ENDPLATE ;
RANGE HISTOGRAM ELECTRON X2 100 0.0 RADIUS  CHKIT
          WINDOW X1 ZMAX_ANODEELECTRODE ZMAX_ENDPLATE ;
```



Range histogram of the electron beam kinetic energy distribution in a 3-MeV diode.

RANGE TRAMLINE Command

Function: Plots transmission line variable as a function of spatial position in a 2D simulation.

Syntax:

RANGE TRAMLINE tline_name variable point, point timer
[options] ;

Arguments:

tline_name	-	name of transmission line, defined in TRAMLINE command.
variable	-	transmission line variable (see TRAMLINE command).
point	-	value of transmission line coordinate (m).
options	-	data-processing options (RANGE [options], Ch. 23).

Description:

The transmission line is specified by tline_name. The variable specifies the particular transmission line variable that can be observed as a function of position, and the range of spatial position is specified by line_name.

(See TRAMLINE command for a list of variables.)

Restriction:

The total number of RANGE commands defined in a simulation is limited to fifty.

See Also:

TIMER, Ch. 11
TRAMLINE, Ch. 13
RANGE [options], Ch. 23

This page is intentionally left blank.

24. 2D and 3D PLOTS

This Chapter covers the following commands:

DISPLAY_2D (2D simulations only)
DISPLAY_ 3D (3D simulations only)
VIEW_ 3D (3D simulations only)
CONTOUR
VECTOR
PHASESPACE
TAGGING

These commands allow you to visualize some selected feature of the simulation over a specified two-dimensional spatial area. The feature might be simulation geometry, a field variable, a particle trajectory, etc. In many commands, the two-dimensional area is specified using an AREA command. In 2D simulations, this area is a portion of the two-dimensional space. In 3D simulations, the area is a portion of a plane; the position and orientation are arbitrary, but the plane must be conformal in one of the coordinates.

You can use DISPLAY_2D and DISPLAY_3D commands to plot the simulation spatial objects and/or spatial grid. The VIEW_3D command provides a three-dimensional perspective view of 3D simulation features. Field variables can be output as CONTOUR plots showing "equipotentials," or they can be output as VECTOR plots showing magnitude and direction. In PERSPECTIVE plots, the variable is plotted in the third axis, thus creating a three-dimensional plot.

You can use the PHASESPACE commands to produce plots of any pair of canonically conjugate momenta. Particle positions are plotted as (unweighted) points in the two-dimensional space. You can also use those commands to obtain trajectory plots, which follow the motion of selected particles as a function of time, and you can select the particles with the TAGGING command.

DISPLAY_2D Command

Function: Displays selected simulation features in a 2D simulation.

Syntax: DISPLAY_ 2D [area] [display_group] ;

Arguments:
area - name of spatial area, defined in AREA command.
display_group - name of geometry display group.
MAXWELL, PERIMETER, OBJECTS, SPATIAL_GRID, MATERIAL_PROPERTIES, OUTER_BOUNDARIES, UNIQUE_GEOMETRIES, and OUTPUT.

Description:

The DISPLAY_2D command provides a visual interpretation of the simulation models you select in a 2D simulation. You may enter any number of DISPLAY_2D commands, and you can vary the content in each plot by selecting different options or by entering the commands at different locations in the command file. (Each DISPLAY_2D command can plot only information available to it from preceding commands, but not from commands which follow it.) In 2D simulations, the default area is the entire spatial area. As an option, you may specify an area to enlarge this area or to zoom in on some selected region.

Entering a DISPLAY_2D command without an area or options automatically displays all of the features of the display groups listed. The default plot area is the entire 2D simulation area. There are seven display groups available. Each group includes some subset of the MAGIC simulation models. The groups are arranged by category of commands. The available display groups and the matching model types are listed in the following table.

Display Group	Includes these Model Types
MAXWELL	OUTER_BOUNDARIES,MATERIAL_PROPERTIES, UNIQUE_GEOMETRIES
PERIMETER	CONDUCTOR, OUTER_BOUNDARIES
OBJECTS	POINT, LINE, AREA
SPATIAL_GRID	MARK, GRID
OUTER_BOUNDARIES	SYMMETRY, PORT, OUTGOING, FREESPACE, MATCH, IMPORT
MATERIAL_PROPERTIES	CONDUCTOR, CONDUCTANCE, DIELECTRIC, EMISSION
OUTPUT	RANGE, OBSERVE
UNIQUE_GEOMETRIES	FOIL, POLARIZER, SHIM, STRUT

If you enter a simple DISPLAY_2D command with an area but with no options, you will automatically display all of the features listed above in the command syntax. If you enter options, then only the selected features will be displayed. For convenience, some options contain subsets of other options. For example, if you enter OBJECTS, the plot will display all points, lines, and areas

The OBJECTS group includes POINT, LINE, AREA, and VOLUME. LINE objects are plotted as thick, red lines. AREA objects are plotted as red cross-hatched regions, and POINT objects are shown with yellow markers.

The SPATIAL_GRID group includes MARK, in which grid markers are plotted as arrowheads on the coordinate axes, and the grid lines plotted as thin, gray lines (every tenth line is dark gray).

The OUTER_BOUNDARIES group contains all of the outer boundaries plotted as dashed lines, SYMMETRY in dark blue, PORT in yellow, etc.

The MATERIAL_PROPERTIES group includes all material properties plotted as cross-hatched lines on area objects, CONDUCTOR in gray, DIELECTRIC in green, CONDUCTANCE in purple, etc.

The UNIQUE_GEOMETRIES group includes all unique geometries plotted as lines; POLARIZER is dashed green lines, STRUT uses purple lines, etc.

The OUTPUT group includes output which requires spatial definition, specifically, RANGE and OBSERVE. It can be used to verify the precise locations of measurements.

The MAXWELL group includes three other groups: OUTER_BOUNDARIES, MATERIAL_PROPERTIES, and UNIQUE_GEOMETRIES. The groups contain all of the features which can affect electromagnetic fields and particle kinematics; these include the IMPORT, PORT, FREESPACE, POLARIZER, and OUTGOING boundaries. In addition, all surface areas which have emission enabled are shown with red-arrow coming from the surface to indicate emission property.

The PERIMETER group includes CONDUCTOR and the OUTER_BOUNDARIES group. Visual inspection of this display should be used to verify that the perimeter is contiguous and closed.

The following table indicates how the models from the various commands are graphically represented.

Model Type	Display Representation
CONDUCTOR	Gray hatched regions outlined in pale cyan.
CONDUCTANCE	Cross-hatched regions, color magenta
DIELECTRIC	Cross-hatched regions, color green.
EMISSION SURFACE	Red arrows directed out of surface.
PORT, MATCH, OUTGOING	Yellow arrows directed to interior of simulation.
IMPORT	Both yellow and red arrows directed to interior.
FOIL, SHIM	Gray hatched region outlined in dark cyan
FREESPACE	Cross hatched regions, color magenta
STRUT	Lines drawn with color magenta
DRIVER	Cross hatched regions, color yellow
POLARIZER	Dark green dashed line.
SYMMETRY	Blue lines drawn with arrows directed to interior
POINT	Points marked with '+', color yellow.
LINE	Lines drawn in dark red.
AREA	Cross-hatched regions, color dark red
GRID	Mesh lines drawn in gray and dark gray every 10^{th} line.
MARK	Black arrows along plot axis indicating fixed points.
OBSERVE	Drawn as bright green regions and marked black points.
RANGE	Lines drawn in green.

Restrictions:

1. There is no restriction on the number of DISPLAY_2D commands.

2. Each plot can include only data from commands which precede the DISPLAY_2D command.

3. All geometry is plotted in true (cartesian) space rather than curvilinear coordinate space.

4. The DISPLAY_2D command is available only in 2D simulations.

Examples:

To help verify simulation integrity, you can display the features which comprise the outer dynamic perimeter. Visual inspection can ensure that there are no "holes" in the outer perimeter, which must be contiguous for the simulation to be valid. In addition, by adding the NOGRID and MARK options you can suppress the display of the finite-difference mesh and have the marked fixed points indicated. Notice in the figure below, that the PORT boundaries on the left and right edge of the simulation geometry are represented by yellow arrow along the port edge. The black arrows along the top and left edge indicate marked fixed points of the grid. The conductors are represented as gray hatched regions outlined in pale blue. The dark blue arrows at the bottom of the display indicate an axial symmetry boundary. The dashed green line indicates the use of the polarizer model. (Note: There is no significance to spacing the line breaks.) Finally, note the red arrows on the right face of the lower conductor. These indicate that the surface is used in a particle emission model. Use the following command to obtain this figure.

```
DISPLAY_2D MAXWELL MARK;
```

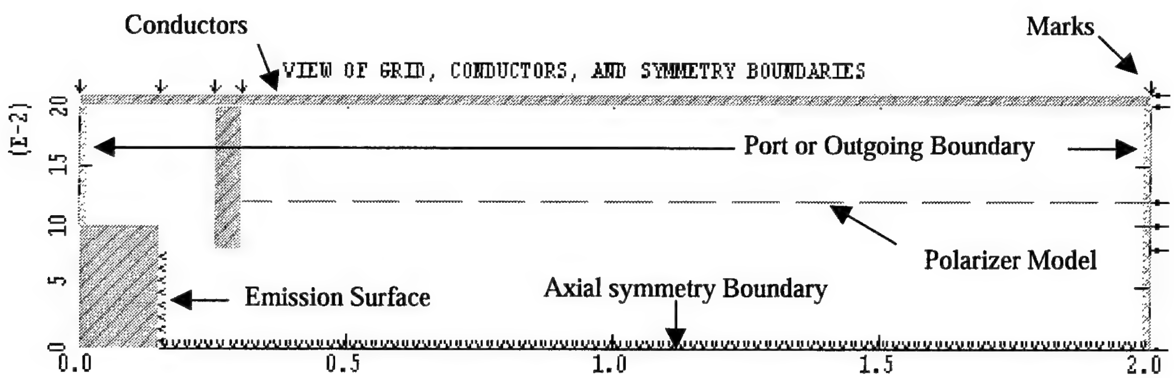


Figure 24-1. Geometry display indicating port boundaries, conductors, helix polarizer model, axial symmetry, and particle emission model. Note that the perimeter is closed. There are no holes.

DISPLAY_3D Command

Function: Displays selected simulation features in a cross-section of a 3D simulation.

Syntax: DISPLAY_3D area [OBJECTS] [SCAN] [SPATIAL_GRID] ;

Arguments: area - name of a conformal spatial area, defined in AREA command.

Description:

The DISPLAY_3D command provides a visual interpretation of simulation features in a cross-section of a 3D simulation. You can enter up to 50 DISPLAY_3D commands. The conformal area is used simply to identify a cross-section or slice through the entire 3D simulation space; a cross-section of the entire simulation is always displayed, irrespective of the actual size of the area. I.e., there is no capability to "zoom" in on a smaller area.

If you enter a simple DISPLAY_3D command with an area but with no options, you will display only the spatial grid. If you enter options, then only the selected features will be displayed. For convenience, some options contain subsets of other options. For example, if you enter OBJECTS, the plot will display all spatial objects which have been assigned properties as OUTER_BOUNDARIES, MATERIAL_PROPERTIES, or UNIQUE_GEOMETRIES. The spatial grid will be imposed on all spatial objects which have been assigned properties, but not on vacuum. The SCAN option is used to view plots of all geometry planes conformal with the specified area. For example, if the area is conformal with the x3 coordinate, then all grid planes in x3 are drawn with the respective geometry. This provides a sequence of slices along the coordinate axis of interest.

Restrictions:

1. Up to 50 DISPLAY_3D commands may be entered.

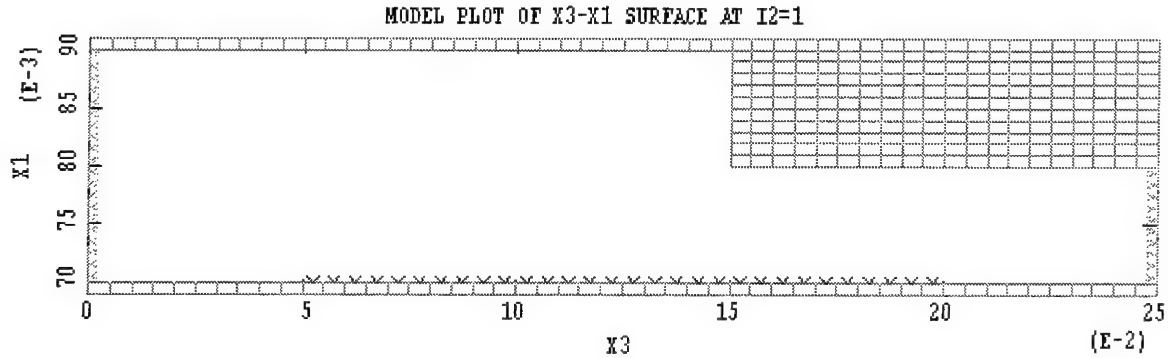
2. All geometry is plotted in true (cartesian) space rather than curvilinear coordinate space.

3. The area specified must be conformal and will be used only to identify the plane; the entire simulation cross section is always displayed, irrespective of the actual size of the area.

4. The DISPLAY_3D command is available only in 3D simulations.

Examples:

The following figure shows the (z,r) cross section of a polar coordinate simulation of a magnetically insulated transmission line. Notice that cells associated with CONDUCTORS are cyan colored rectangles, the PORT boundaries are drawn with yellow vectors pointing to the simulation interior, and the conductor cells which are enabled for EMISSION are drawn with red vectors, pointing into the vacuum.



This figure shows cross section of MITL geometry, indicates the presence of cells where emission can occur, and shows the port boundaries where electromagnetic waves can enter and escape.

VIEW_3D Command

Function: Displays three-dimensional perspective of selected features in a 3D simulation.

Syntax:

```
VIEW _3D [ volume ]
        [ { CONDUCTOR, CONDUCTANCE, DIELECTRIC, VACUUM } ]
        [ EYE_POINT eye_point ]
        [ FOCAL_POINT focal_point ]
        [ SKY_POINT sky_point ]
        [ MAGNIFICATION factor ]
        [ MOVIE {X,Y,Z} [ROTATION start_angle end-angle step_angle] [ZOOM factor]] ;
```

Arguments:

volume	- name of spatial volume, defined in VOLUME command.
eye_point	- spatial point representing observer's eye.
focal_point	- focal point near or within the spatial volume.
sky_point	- spatial point representing vertical axis of the display plot.
factor	- magnification factor (default = unity).

start_angle,stop_angle,step_angle – rotation angle limits for movie option.

Defaults:

The following table lists the default values used for unspecified arguments.

Keyword	Arguments	Default Value
-	volume	entire simulation
{ CONDUCTOR, CONDUCTANCE, ... }	-	CONDUCTOR
{SURFACE_GRID, WIRE_FRAME}	-	SURFACE_GRID
EYE_POINT	eye_point	code-calculated
FOCAL_POINT	focal_point	code-calculated
SKY_POINT	sky_point	code-calculated
MAGNIFICATION	factor	unity

Description:

The VIEW_3D command provides a three-dimensional perspective plot of specified simulation features. You can enter up to 50 VIEW_3D commands, and you can vary the content in each plot by selecting different options.

The plot will be restricted to the specified volume. If no volume is specified, the entire simulation volume will be plotted. You may specify a material property (CONDUCTOR, CONDUCTANCE, or DIELECTRIC) or VACUUM to identify the objects to be plotted. The default is CONDUCTOR. If VACUUM is specified, the plot will represent vacuum as a solid.

The EYE_POINT, FOCAL_POINT, and SKY_POINT options allow you to observe the specified simulation feature from any desired distance and orientation. The eye_point, focal_point, and sky_point represent positions of the observer's eyeball, the central location on the spatial object, and a location which defines the vertical plot axis, respectively. If the size of the image on the screen or page needs to be changed, use the MAGNIFICATION factor to increase or decrease the size.

The MOVIE option allows you to create a sequence of bitmaps in which the perspective view of the geometry is rotated about one of the three cartesian axes X,Y, or Z. If no rotation limits are specified, a sequence of 120 bitmaps is created as the perspective is rotated through 360 degrees. You may simultaneously apply a zoom factor which will gradually zoom in or out depending upon the value of the factor. The view will zoom in for factor>1, and will zoom out for factor <1. Note: The application of the zoom factor is in addition to the magnification factor, which is the starting magnification of the perspective view.

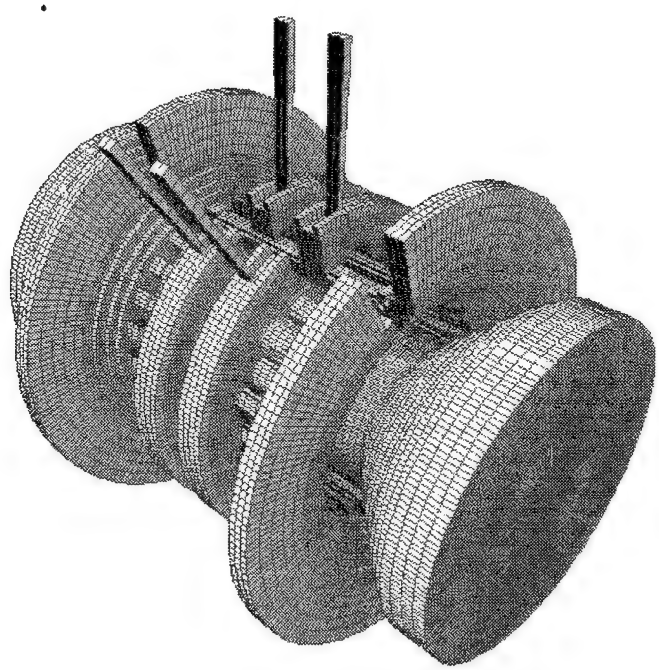
Restrictions:

1. Up to 50 VIEW_3D commands may be entered.
2. All geometry is plotted in true (cartesian) space rather than curvilinear coordinate space.
3. The eye_point must lie outside of the specified spatial volume.
4. The VIEW_3D command is available only in 3D simulations.

Example:

In the following command, the MOVIE option allows you to create a sequence of bitmaps in which the perspective view rotates about the Y axis by 5 degrees per bitmap. Notice that the initial magnification is set to 1.5 times the default, and that the zoom option will cause the view to pull away from the figure.

```
   VIEW_3D CONDUCTOR MOVIE Y ROTATION 0DEGREE 360DEGREE 5DEGREE ZOOM 0.5
MAGNIF 1.5;
```



Perspective view of three dimension simulation geometry. Shows only the conductor geometry. The confining shell has been removed.

CONTOUR Command

Function: Specifies contour plot of electromagnetic field variable.

Syntax:

```
CONTOUR FIELD field area timer
        [ MOVIE ]
        [ AXIS {X, Y, Z} minimum maximum [ step ] ]
        [ TRANSFORM f(x,y,z) ]
        [ INTEGRATE { X, Y } {POSITIVE, NEGATIVE} ]
        [ { NOSHADE, SHADE, BLOCKSHADE } ]
        [ { NOLEGEND, LEGEND } ]
        [ { NODUMP, DUMP } ]
        [ { PLOT, NOPLOT } ] ;
```

Arguments:

field - field variable (E1, E2, ...).
area - name of spatial area, defined in AREA command.
timer - name of timer, defined in TIMER command.
minimum,... - axis boundaries (real).
step - axis sub-division size (real).
f - transformation function, defined in FUNCTION command.
x,y - dummy arguments representing spatial coordinates.
z - dummy argument representing the field variable.

Defaults:

The following table lists the default values used for unspecified arguments.

Keyword	Arguments	Default Value
AXIS	minimum, maximum, step	code-calculated, ten contours
{ NOSHADE, SHADE,BLOCKSHADE }	-	NOSHADE
{ NOLEGEND, LEGEND }	-	NOLEGEND
{ NODUMP, DUMP }	-	NODUMP
{ PLOT, NOPLOT }		PLOT

Description:

The CONTOUR command is used to plot contours of an electromagnetic field variable over a specified two-dimensional spatial area. Each contour curve shows locations where the field has the same magnitude. Positive-valued contours are drawn with solid lines, and negative-valued contours are drawn with dotted lines. Plots are produced when specified by the timer.

The MOVIE option permits you to generate a sequence of bitmaps (in PCX format) from a sequence of RANGE plots. In order for this sequence to provide an appealing visual representation of the data, you should also use the AXIS option to enforce limits. When the MOVIE option is invoked, MAGIC generates a series of bitmaps that are saved in a folder. There is one movie folder per command that uses the MOVIE option. You can "play" the movie by using the MOVIE command. Alternatively, you can post process the files to create an AVI file.

The AXIS option can be used to re-define the plot limits both in the spatial axes (X and Y) and in the field variable axis (Z). (By default, the spatial axis boundaries depend on the specified area, and the spatial grid

and the field axis boundaries depend on the extrema; values are printed only at the endpoints of the axes.) The minimum and maximum specify axis boundaries, and the optional step determines axis sub-divisions and printed values. Since non-Cartesian results are converted to present, true, physical perspective, the units for X and Y are always meters. Use AXIS Z to enter maximum and minimum contour values and the step size between contours.

The TRANSFORM and INTEGRATE options are the two data-processing operations which can be applied to the contour data; by using the TRANSFORM option, contours of the function (f) will be plotted instead of the field variable (z). By using the INTEGRATE option, the integral of the contour data can be taken along one coordinate (X or Y) in either the POSITIVE or NEGATIVE direction. This option is useful in certain cases for illustrating quasi-static electric or magnetic potentials.

The SHADE option can be used to shade areas between the contours. The LEGEND option can be used to create a legend box on the right-hand side of the contour plot, which lists the values represented by the color-coded contours. The BLOCKSHADE option selects cell shading, in which each cell is shaded by its mean value.

The DUMP option can be used to divert contour plot data to the FLD file for post-processing. In this event, NOPLOT will prevent plotting this data at the end of the simulation.

Restrictions:

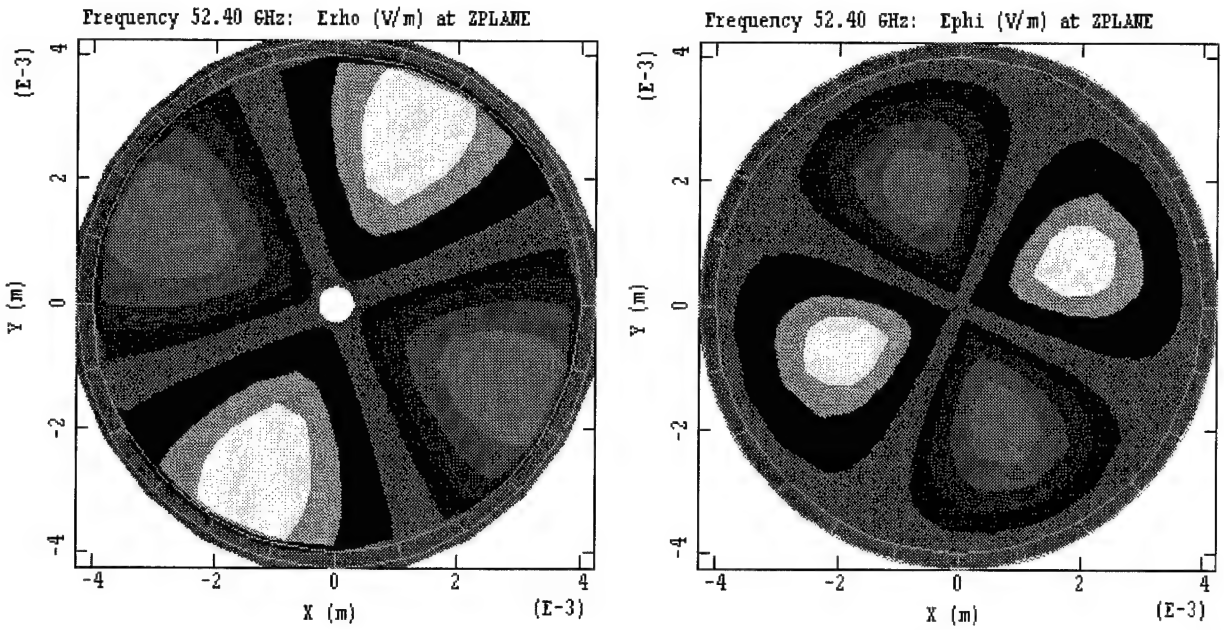
1. The total number of CONTOUR commands is limited to twenty.
2. If a contour step is entered, it must be sufficiently large that no more than twelve contours will result.

See Also: **FUNCTION, Ch. 6**
TIMER, Ch. 11
GRAPHICS. Ch. 20

Examples:

In this example, the eigenmode solutions of a cylindrical pillbox of width = 0.39315 cm and radius of 0.400 cm are desired. Contours are requested in the mid-plane of the pillbox, and the SHADE option is used.

```
CONTOUR FIELD E1 ZPLANE TSYS$EIGENMODE SHADE;
CONTOUR FIELD E2 ZPLANE TSYS$EIGENMODE SHADE;
```



The figure on the left shows the E_r field for an n=4 mode, and the figure on the right shows the E_ϕ component for this same mode. Notice that the simulation frequency is 52.4 GHz.

VECTOR Command

Function: Specifies vector plot of electric, magnetic, or current-density field vectors.

Syntax:

```
VECTOR  FIELD  field, field  area  timer
        [ MOVIE ]
        [ AXIS {X, Y}  minimum  maximum  [ step ]
        [ DENSITY  x_vectors  y_vectors ]
        [ { LINEAR, LOGARITHMIC [ decades ]  } ]
        [ { NODUMP, DUMP } ]
        [ NORMALIZATION  magnitude ] ;
```

Arguments:

field, field	- field variables for x and y components of vector (E1, E2, ...).
area	- name of spatial area, defined in AREA command.
timer	- name of timer, defined in TIMER command.
minimum,...	- axis boundaries (m).
step	- axis sub-division size (m).
x_vectors, y_	- number of vectors along x and y axes (integers).
decades	- number of decades in logarithmic scaling (integer).
magnitude	- maximum magnitude for vector (real).

Defaults:

The following table lists the default values used for unspecified arguments.

Keyword	Argument	Default Value
AXIS	minimum, maximum, step	code-calculated
DENSITY	x_vectors, y_vectors	30, 30
{LINEAR, LOGARITHMIC}	-	LINEAR
NORMALIZATION	magnitude	code-calculated

Description:

The VECTOR command is used to plot a set of arrows which represent electromagnetic field vectors over a specified two-dimensional spatial area. Each arrow shows the relative magnitude and direction of the field vector at that location in space.

Two field variables (e.g., E1 and E2) must be entered to completely define the field vector. The choices are limited to electric fields, magnetic fields, or current-density fields. They cannot be co-mingled (e.g., E1, J3 is not legal). Furthermore, the vectors are plotted over a specified spatial area, and the vectors must lie in the plane of the area. Thus, the vector components also define the spatial axes. For example, if the area is a plane in X1, X2 coordinates (conformal in X3), then you can plot E1 and E2, but not E1 and E3. If the simulation is performed in a non-cartesian coordinate system (SYSTEM, Ch. 10), the results will automatically be transformed to provide true, physical perspective. Plots are produced when specified by the timer.

The MOVIE option permits you to generate a sequence of bitmaps (in PCX format) from a sequence of RANGE plots. In order for this sequence to provide an appealing visual representation of the data, you should also use the AXIS option to enforce limits. When the MOVIE option is invoked, MAGIC generates a series of bitmaps that are saved in a folder. There is one movie folder per command that uses the MOVIE option. You can "play" the movie by using the MOVIE command. Alternatively, you can post process the files to create an AVI file.

The AXIS option can be used to re-define the plot limits in the spatial axes (X and Y). The minimum and maximum specify axis boundaries, and the optional step determines axis subdivisions and printed values. Since non-cartesian results are converted to present, true, physical perspective, the units for X and Y are always meters. (By default, the spatial axis boundaries depend on the area and the spatial grid; only the axis end points have values printed.)

The DENSITYoption is used to select the number of vectors to plot along each axis. The SCALE option is used to select linear or logarithmic scaling. With logarithmic scaling, the length of the arrows will be proportional to the logarithm of the field magnitude, rather than the magnitude itself. The decades are basically a cutoff to avoid plotting low-magnitude vectors. A particularly useful choice is LOGARITHMIC 100. Since most field magnitudes are only a few decades below maximum, all plotted vectors will have nearly the same maximum length, producing a vector plot displaying only direction.

Restrictions:

1. The total number of VECTOR commands is limited to twenty.

2. The vectors must be either electric fields, magnetic fields, or current-density fields. They cannot be co-mingled.

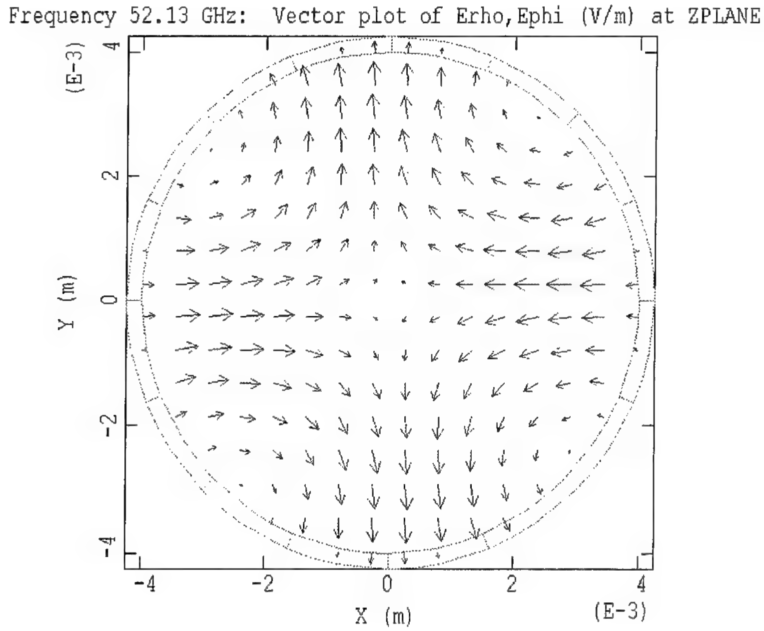
3. The area must be planar, and the field vectors must lie in the plane.

See Also: **SYSTEM, Ch. 10**
TIMER, Ch. 11

Examples:

The following figure comes from a 3D eigenmode simulation of a pillbox cavity.

```
VECTOR FIELD E2 E3 ZPLANE TSYS$EIGENMODE ;
```



This figure shows the electric field vectors for a particular eigenmode of a pillbox cavity cut across the axis of symmetry.

PHASESPACE Command

Function: Plots particle phase space.

Syntax:

```
PHASESPACE  AXES  coordinate, coordinate  timer
        [ MOVIE ]
        [ AXIS {X, Y}  minimum  maximum  [ step ] ]
        [ WINDOW  coordinate  minimum  maximum ]
        [ SPECIES  species ]
        [ { NOTAG, TAG } ]
        [ { NODUMP,  DUMP }]
        [ { PLOT, NOPLOT } ] ;
```

Arguments:

coordinate,... - coordinates for the X and Y axes and optional window (X1, X2, X3, P1, P2, P3, Q, KE, GAMMA, or f(x1,x2,x3,p1,p2,p3,q,ke), a function defined in a FU\NCTION command).

timer - timer name, defined in TIMER command.

species - particle species (ALL, ELECTRON, PROTON, or defined in SPECIES command).

Defaults:

The default values for the optional arguments are listed in the Table.

Keyword	Argument	Default Value
AXIS	minimum, maximum, step	(see Description, below)
WINDOW	variable	(infinite window)
SPECIES	Species	ALL
{ NOTAG, TAG }	N.A.	NOTAG

Description:

The PHASESPACE command produces plots of particle phase space at times specified by the timer. The timer must specify times that take into account the kinematics step_multiple (LORENTZ, Ch. 18). To obtain trajectory plots (phase-space with a time duration), use the INTEGRATE option in the timer (trajectory plots automatically use only tagged particles).

The coordinates specify the phase-space variables. The standard variables that may be specified are X1, X2, P1, P2, or P3. These are the physical coordinates and momenta of the particles. (Figure 24-6 presents a classic phase-space result.) In addition, the arguments, Q, KE, and GAMMA may be used to select the charge per macroparticle (coulombs), kinetic energy per physical particle (eV), and relativistic factor (unitless). Other variables can be selected for the axes by using a function name for the arguments, axaxis and ayaxis. The function implicitly refers to seven particle variables: x1, x2, x3, p1, p2, p3, q, and ke. The variables x1, x2, and x3 denote the physical coordinates of the particle. The variables, p1, p2, and p3, denote the momenta of the particle in m/s. The variable q denotes the particle charge, and the variable ke denotes the kinetic energy in eV.

The MOVIE option permits you to generate a sequence of bitmaps (in PCX format) from a sequence of RANGE plots. In order for this sequence to provide an appealing visual representation of the data, you should also use the AXIS option to enforce limits. When the MOVIE option is invoked, MAGIC generates a series of bitmaps that are saved in a folder. There is one movie folder per command that uses the MOVIE option. You

can "play" the movie by using the MOVIE command. Alternatively, you can post process the files to create an AVI file.

The keywords, AXIS, WINDOW, and SPECIES are used to set optional parameters. These can be entered in any order after the AXES keyword and parameters are specified.

The keyword, AXIS, is used to establish the plot axes limits. The arguments, rmin and rmax, specify the axis extrema. The argument, rstep, specifies the step size used in labeling the plot axis. A value of zero for rstep causes only the end points of the axis to be labeled. Not entering a value has the same effect.

The keyword, WINDOW, can be used to specify an acceptance window in any of these variables: x_1, x_2, p_1, p_2 and p_3. Only particles falling within the variable limits, wmin and wmax, will be plotted. Note that the window variable does not have to be one of the plotted variables; it simply offers a mechanism to discriminate plotted particles.

The keyword, SPECIES, is used to select the particle species to be included in the plot. The argument, ALL, will produce a plot with all species on one plot. Plots of individual species may be created by specifying the particular species name, e.g., ELECTRON, PROTON, etc. The default for species is ALL. The NOTAG, TAG option allows you to select all particles or tagged particles only (TAGGING, Ch. 24). The default is NOTAG (all particles).

The DUMP option can be used to divert contour plot data to the FLD file for post-processing. In this event, NOPLOT will prevent plotting this data at the end of the simulation.

Restriction:

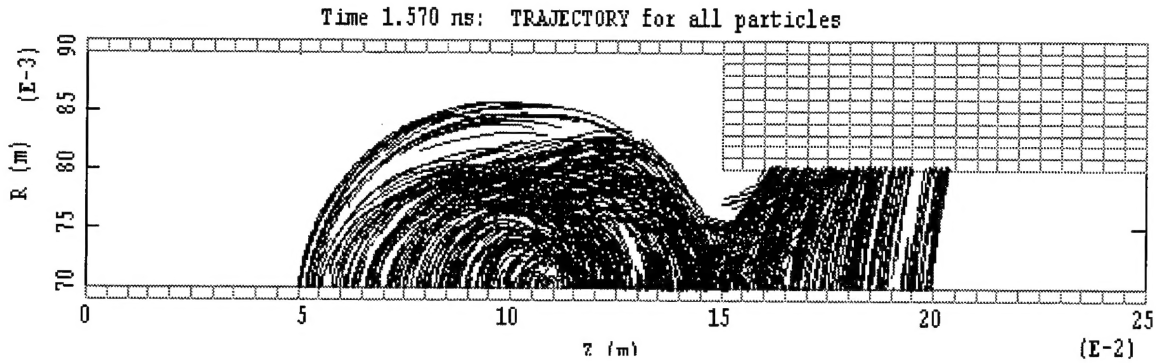
The total number of PHASESPACE commands is limited to 20.

See Also: **TIMER, Ch. 11**
SPECIES, Ch. 18
TAGGING, Ch. 24

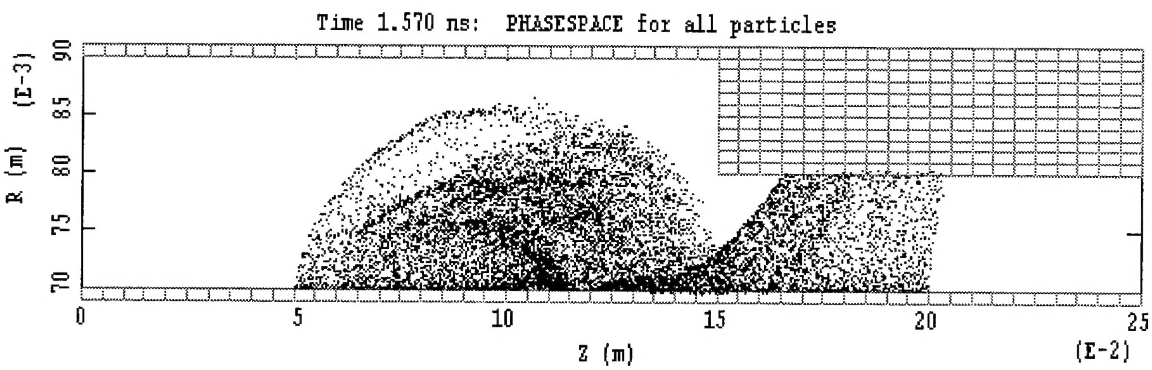
Examples:

1. The following commands produce phase-space plots at time step 600. The first plot will be of the position (z,r) phase space, and the second plot will be a trajectory plot of the tagged particles spanning 50 time steps.

```
TIMER For_Phase PERIODIC INTEGER 600 99999 500 ;
TIMER For_Traj  PERIODIC INTEGER 600 99999 500 INTEG 50;
PHASESPACE AXES  X3 X2  For_Phase ;
TAGGING 0.05 ;
PHASESPACE AXES  X3 X2  For_Traj ;
```



This figure shows electron particle trajectories of the 5% tagged electrons. The trajectory path spans 50 time steps, ending at time 1.57 ns.



This figure shows the instantaneous electron phase space at 1.57 ns.

TAGGING Command

Function: Reduces number of particles plotted.

Syntax: TAGGING fraction (t) ;

Arguments: fraction - fraction of particles tagged ($0 < \text{fraction} \leq 1$), constant or function defined in a FUNCTION command.

Default:

The default tagging fraction is unity.

Description:

The TAGGING command specifies the fraction of particles chosen to appear in particle plots. By default, the fraction is 1.0, and particle plots will show all particles in the simulation. However, this default may produce an extremely dense plot and an extremely large plot file. Setting fraction to a value less than unity limits the plots to a randomly chosen fraction of the particles. Note that individual output commands, e.g., PHASESPACE, may offer an option to plot either all the particles or the tagged particles only.

Restriction:

Tagging affects particles of all species.

See Also: **TABLE PARTICLES, Ch. 21**
PHASESPACE, Ch. 24

Examples:

The following example illustrates the use of TAGGING in conjunction with DUMP and PHASESPACE to create a data file which can be post processed to make a movie. The TAGGING fraction is set to .25 to prevent the data file from becoming too large. The DUMP command is used to store the phase-space data in the data file. The TIMER command is used to specify the rate at which the phase-space data is dumped, and the PHASESPACE command is used to select the specific phase-space data. The commands are:

```
TAGGING 0.25 ;
DUMP TYPE PHASESPACE NOPLOT ;
TIMER For_Movie PERIODIC INTEGER 1 99999 8 ;
PHASESPACE AXES X1 X2 For_Movie TAG ;
```